HOW TO USE YOUR RADIO SHACK PRINTER

WILLIAM BARDEN, JR.

INCLUDES ALL CURRENT PRINTERS

CGP 115
220

DMP 100
110
120
200
400
420
500
2100
2100P

DW I, II,
IIB

DWP 210
410

LP I, II,
III, IV,
V, VI,
VII, VIII

QP I, II

TP 10

Plotter/Printer

TANDY®

A MICROTREND/VALLEYWARE BOOK

How To Use Your Radio Shack Printer

William Barden, Jr.

A Microtrend/Valleyware Book
San Diego

Published in 1985 by Microtrend
3719 Sixth Avenue
San Diego, CA 92103

First Edition
First Printing — 1985
1 2 3 4 5 6 7 8 9

Interior Design by Ed Roxburgh and Richard Carter

Library of Congress Catalog Card No. 85-00000
ISBN 0-915391-09-0

Printed in the United States of America

Contents

PREFACE

Printers for computers have gotten too smart! Only a few years ago when you bought a printer for a computer system, you could print lines, and possibly underline text, and that was about it. Now, however, you can print in boldface, print graphics patterns to draw figures, print proportionally spaced text similar in appearance to typesetting, vary the line and character spacing, and create your own character fonts! The only trouble is, you must know how to tell your printer to do these wonderful things. And that's what this book is about—to let you know how to use your Radio Shack printer to its best advantage.

You can use this book for any Radio Shack computer—Model I, II, III, 4, 4P, 12, 16, Tandy 2000, Model 100, and MC-10. The common element here is your system printer—it operates fairly independently of the type of computer you're using.

This book is for the new generation of Radio Shack printers, the ones that have become so intelligent that it's sometimes confusing to understand what's involved in utilizing all of their features. We'll try to unravel the printer "codes" in this text and give you simple instructions on doing things such as:

- Spacing lines and characters
- Performing bold printing and underlining
- Superscripting and subscripting
- Printing graphics patterns and pictures
- Plotting graphs
- "Dumping" the display in either text or graphics modes
- Playing typesetter to produce your own letterheads or logos

Printers are not hard to use once you understand some of the concepts involved. In the first section of this book, we'll give you some of the background on printers. Regardless of the printer you have, all printers do about the same types of things in about the same way. Knowing some of the basics greatly simplifies your use of Radio Shack printers whether you have the simplest printer or the most complex.

In Section 1, we'll describe in general terms what can be done with your printer, what printer terms you should be familiar with, the way the printer

communicates with your computer system, the "character set" found in all printers, and the types of printing that you can do. We'll also discuss Radio Shack printer codes and how they evolved.

In Section 2, we'll cover specific text-related functions that can be performed by your printer—everything from listing programs and data, to proportional spacing. In doing this we'll provide plenty of concrete examples that can be used for any Radio Shack printer. These examples will all be in Radio Shack BASIC, but you can apply them to assembly language or other languages as well. This section covers text printing only and applies for every model of printer, from an early Line Printer I to the new DMP-2100 and daisy-wheel models.

In Section 3 of the book, we'll provide information on how to use the graphic functions of your printer. There's a lot of material covered here—everything from simple plotting to complete "high-resolution" graphics pictures and screen dumps. Graphics is one of the areas in which not much information is available, and we've tried to give you many practical examples of graphics applications. Although the section primarily covers dot-matrix printers, it also describes how to plot points and graphs with daisy-wheel printers.

The appendix is a list of decimal, binary, and hexadecimal conversions from 0 through 255, which should help in preparing printer data.

Want to use the hidden power of your smart printer that you've been unable to tap? Try the examples here and we guarantee you'll be able to do more than just print lines. Now where did I put that replica of the Magna Carta that I had printed out on a DMP-2100 . . .

WTB, jr.

This book is dedicated to those unsung frontiersmen of the West—the UPS people who delivered dozens of printers to my computer rooms. Sorry about that Radio Shack beach ball on the stairs, Tiny . . . .

SECTION 1

Printer Basics

CHAPTER 1

A BRIEF LOOK AT RADIO SHACK PRINTERS

In this chapter we'll look at how microcomputer printers have evolved over the last few years. We'll see how printers have become the intelligent devices they are today—actually self-contained computers in their own right. We'll also look at which printers are covered in this book, and describe in general terms what each is capable of doing.

HOW RADIO SHACK PRINTERS EVOLVED INTO INTELLIGENT DEVICES

Early Radio Shack printers, and the printers of all other manufacturers, had fewer capabilities than ordinary typewriters. Generally, they could print only lines of text in a single text style. The spacing for the lines was usually set at six lines per vertical inch, and the spacing between characters was usually ten characters per inch along the line. No underlining or boldface was allowed. Printing produced by such printers looked very similar to what you see in Figure 1-1.

This is a sample of text produced by the new Radio Shack dot-matrix printer.
This breakthrough in small computer equipment design forms characters by
printing a matrix of dense dots. Special features of the new printer include
UPPER CASE characters, lower case characters, periods, and commas.

Figure 1-1. Ancient (1970) Printing in Computer Printers

Contrast the quality of this printing with the printing of a moderately priced Radio Shack printer of today, shown in Figure 1-2. Quite a difference!

In 1978, the printing in Figure 1-1 was about all that *any* computer printer would do. Even printers for larger computer systems generally just spewed out simple, uncomplicated lines of text, although at some very rapid rates.

(The rates were so fast that an unattended printer might literally eject paper halfway across the room!)

Three elements contributed to the growth of printers from machines that simply printed text to machines that performed all kinds of fancy functions—the microprocessor, word processing, and "high-resolution" graphics.

Printer Hint

WHAT PRINTERS DID IN THE '70s

Printers in the '70s generally just printed out pages and pages of reports and program listings. The printed material was of fairly high-quality, but not oriented towards word processing. Typical printers were high-cost chain and drum printers, although there were dot-matrix printers at the low end of the price scale—about $2,000 for a minicomputer printer.

MICROPROCESSORS ARE PRINTER CONTROLLERS

Microprocessors are "computers on a chip" that literally reduce the size of a great deal of complicated electronics circuitry to a "chip" about the size of a stick of chewing gum. Every Radio Shack computer—the Color Computer, Model 4, Tandy 1200 and 2000, and others—has a microprocessor as its central computing component.

Microprocessors, though, are used in more than general-purpose microcomputers. They were originally designed to be used in peripheral devices such as video terminals. Oddly enough, microprocessors really weren't employed in these devices in any great quantities until after inexpensive microcomputers (such as the Radio Shack Model I) became a reality. Design engineers then started using microprocessors not only in microcomputers, but also in devices that attached to microcomputers. One of the first uses of microprocessors in peripheral devices was in microcomputer printers.

This is a sample of text produced by the Radio Shack DMP-2100. This **printer** is a **dot-matrix** printer, but it has a 24-wire printhead, **enabling near** letter-quality type. Special features of this new **printer** include <u>underlining</u>, compressed and expanded printing, **correspondence** or standard printing, **bold printing**, proportional spacing, and many others.

Figure 1-2. Contemporary Computer Printing

Microprocessors operate at such high rates of speed in making decisions that they can easily handle the relatively slow speed of a mechanical device, such as a line printer. The microprocessor, through a microprocessor program, makes such decisions as Are we at the end of a line?, How many characters have been printed?, and Have we spaced to a new line?.

At first a *little* intelligence crept into printers. Line printers typically printed a line and then did a carriage return back to the beginning of a new line in the same way that a typewriter does a carriage return. One of the earliest features to be put into printers was the ability to do "bidirectional printing." The time during which the carriage returned was used to print the next line or a portion of the next line in reverse, speeding up the overall printing speed, as shown in Figure 1-3.

More and more intelligence was installed in printers as time went on. Current printers all have microprocessors built in and are small computers in themselves. The microprocessor interprets the stream of characters coming from the microcomputer and controls the actions of the printer. Microprocessors control the spacing for proportional printing, graphics printing, and boldface printing. And there's more to come. With each generation of printers, look for more and more intelligence to be built in!

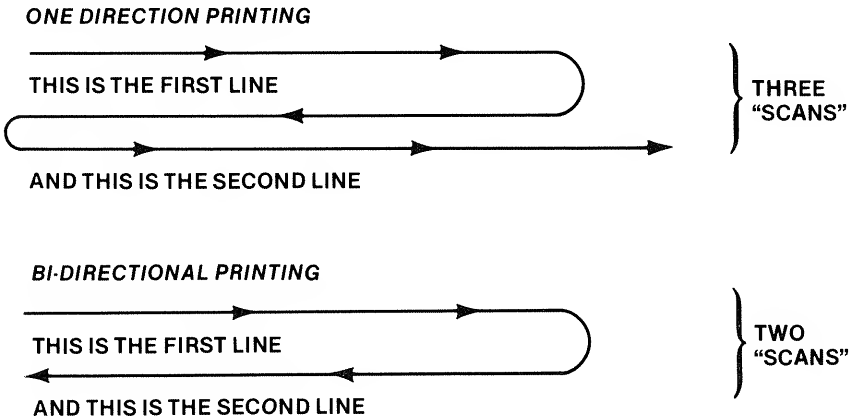


Figure 1-3. Bidirectional Printing

WORD PROCESSING

The second factor affecting printer capabilities was the rise in importance of word processing. One of the first applications for which microcomputers were used was processing text material. Computer users weren't happy with simply *printing* text—they wanted the ability to underline, print in bold, print superscripts and subscripts, and otherwise enhance the appearance of printed documents.

To implement these functions, more sophisticated printers evolved, using the built-in microprocessor and controlling computer programs. Current Radio Shack printers can produce documents that look almost as if they had been typeset. The printers have many functions related to word processing. The Radio Shack DMP-2100, for example, produces near-letter-quality type in a variety of "fonts." The future holds the promise of many more features in printers that will enable a user to produce documents that meet or exceed the typesetting standard.

GRAPHICS CAPABILITIES

The third factor that has prompted development of printers is greater graphics capabilities in microcomputers. The Color Computer can display 256 by 192 picture elements and the Tandy 2000 can display 640 by 400 picture elements on its monitor. In both cases, as well as in other microcomputers, it's handy to be able to reproduce the screen on a printing device. Current Radio Shack printers fulfill this need, in addition to generating plots and graphs without first going to the computer screen. Again, the future will be more and more elaborate graphics capability in printers. The Radio Shack DMP-2100 prints 180 separate points per vertical inch and up to 300 points per horizontal inch with a 24-wire print head, and there's every indication that this print density will be exceeded in future printers.

WHAT YOUR PRINTER CAN DO

All in all, then, your Radio Shack printer is a fairly smart device that can do much more than simply print lines. We'll have to qualify this, however, and say that what your printer can do is pretty much related to its age. If you have an older Radio Shack printer, you won't be able to accomplish as

Printer Hints

HOW MANY POINTS ARE ADEQUATE?

Obviously, you can never have too many points available for printer graphics. A 35-millimeter film negative has about 3000 by 3000 points—a total of 9,000,000 separate elements—or about 2,200 points per linear inch, and some people still complain about graininess. Even the density of the DMP-2100 at 180 points per inch, is not good enough to define typefaces as precisely as those produced by typesetters, which use about 2500 scan lines per inch to construct characters. However, there's another side to having a lot of points—it takes scads and scads of time for the computer to process the points. Those 9,000,000 elements of the 35-mm negative would take 25 hours of processing time in BASIC before the graphics picture could be printed out!

much as you could with some of the current models. To give you some idea of what your printer is capable of and what we'll be covering in the last two sections of this book, here's a thumbnail sketch of each of the printers covered in this book:

DOT-MATRIX PRINTERS USING NORMAL PAPER

LPI, LPII: These are the first Radio Shack printers and, like other printers of the time, are not very "intelligent." You'll find very little that can be done with these models. We'd strongly advise buying a newer printer—even the inexpensive ones will far exceed the capabilities of the LPI and LPII.

LPIII: This is the next Radio Shack printer after the LPI and LPII. Although it's a fast printer and will handle larger paper widths, it's also not very smart. If your primary goal is good copy and graphics, and not print speed, think seriously about trading up to a more recent printer.

LPIV: This printer contains more intelligence than the LPIII and can print in a number of different type styles. Its primary goal was to produce proportionally spaced text. It has no graphics capability.

LPV: This is basically an upgrade of the LPIII with bold printing and underlining capability. A good, high-speed printer, but useful primarily for straight text.

LPVI: A good, high-speed printer that can handle larger paper widths, but without many other features.

LPVII: This is one of the first graphics printers that could print dot patterns. It's weak on text printing, but does a fairly good job on graphics.

LPVIII: This is one of the first Radio Shack printers to offer not only graphics, but word processing capability. It offers a variety of features such as underlining, subscripting, block graphics, and a number of different type styles.

DMP-100: This is an upgraded Line Printer VII, with underlining capability. It's primarily a graphics printer without word processing frills.

DMP-200, DMP-400, DMP-500: These three printers are upgrades of the LPVIII, LPVI, and LPV, respectively. They all offer both word processing and graphics modes. In general, they aren't as powerful as current Radio Shack printers, but are fairly flexible.

DMP-110, DMP-120, DMP-420: These are some of the most recent Radio Shack printers and offer a wide range of both word processing and graphics features.

DMP-2100: This is the top-of-the-line dot-matrix printer that has a wide range of word processing and graphics features. The characters are formed by a high-density print head to produce text approaching daisy-wheel quality.

DOT-MATRIX PRINTERS USING SPECIAL PAPER

Quick Printer I and II. These are early, inexpensive printers using special thermal paper. Neither has special features nor graphics capability. Best to trade up on these if you want to do anything significant.

TP-10: This is a recent addition to the Radio Shack line, designed for the MC-10 computer, although it can be used on the Color Computer. It will print plain text and low-density graphics when used with the MC-10 or Color Computer. Not too powerful, but inexpensive.

DAISY-WHEEL PRINTERS

DW-II: This is an "impact" (non dot-matrix) printer with interchangeable character wheels. It's a top-of-the-line word processing printer without graphics capability in the usual sense, although graphics can be done by printing periods or other characters. The DW-IIB is a more recent version.

DWP-410: This is a slower, less expensive version of the DW-II.

DWP-210: This is a still slower, less expensive version of the daisy-wheel printer.

Qume, WP-50: These are early, discontinued daisy-wheel printers.

COLOR GRAPHICS PLOTTERS

CGP-115: This is an inexpensive four-color plotter device that can also print text characters. Graphics is done by a different method than the dot-matrix printers. We'll cover only its text character modes in this book.

CGP-220: This is a color ink-jet printer that is essentially a dot-matrix printer with limited word processing capability.

WHAT THIS BOOK WILL COVER

You can see that there have been quite a few models of printers with more and more built-in intelligence as the models progressed. Current Radio Shack printers conform to a standard set of printer codes—the same codes are used to print an underline on a DMP-110 as on a DWP-420, for example. We'll use these codes as a base in showing you how to use your printer to perform a variety of functions from proportional spacing to graphics pictures. We'll also note which printers cannot perform a function as we describe it. In some cases, there are alternative ways to perform the actions, and we'll show you what can be done with your printer if it's an earlier model.

This book is basically divided into three separate sections. In the section you're reading we'll give you the background on Radio Shack printer specifications and define some of the terms used in the remainder of the book. If you're not very familiar with printers, it'll probably be worth your while to read through this section. It's fairly easy material.

In the second section we'll show you how to implement different text and word processing functions with your printer. This will apply to all printers, because every printer has the capability to print lines of text characters. In the third section we'll show you how to perform graphics functions on your printer, assuming, of course, that you have a later model printer with graphics capability. We'll also show you how to use your daisy-wheel printer for limited graphics.

What will you need to know to use the material in this book? Some knowledge of BASIC will be a great help. If you don't have a good knowledge of BASIC, however, there are still a great many things that can be done without writing involved, complex BASIC programs. One example is constructing a five-line program to print mailing labels on your printer or an equally short program to produce your company's logo. We'll show you how as we go along.

Printer Hint

WHY SO MANY PRINTERS?

Confused by too many Radio Shack printers? You should count your blessings. Radio Shack has continued to bring out printers at the rate of about four per year since they first entered the small computer business. The significant thing about new printers is that they keep pushing the technology—more and more intelligence, higher-density graphics, and better-looking type faces. And lower prices for comparable features. If I sound like a Radio Shack shill, I don't mean to—it's just that printers are one area in which there are still a lot of improvements to be made. I'm looking forward to next season's models!

CHAPTER 2

BASICS OF PRINTING

In this chapter we'll take a look at the different types *of printers and the different* modes *of printing. We'll then describe some of the basic concepts in printing both text and graphics.*

TYPES OF PRINTERS

There are five different types of printers that Radio Shack currently carries: dot-matrix printers, daisy-wheel printers, thermal printers, ink-jet printers, and printer/plotters. Each printer really accomplishes the same task—producing text or a graphics pattern on a piece of paper. In most cases, the printed characters will be a black or contrasting color on a paper background. In a few cases, colored printing is produced.

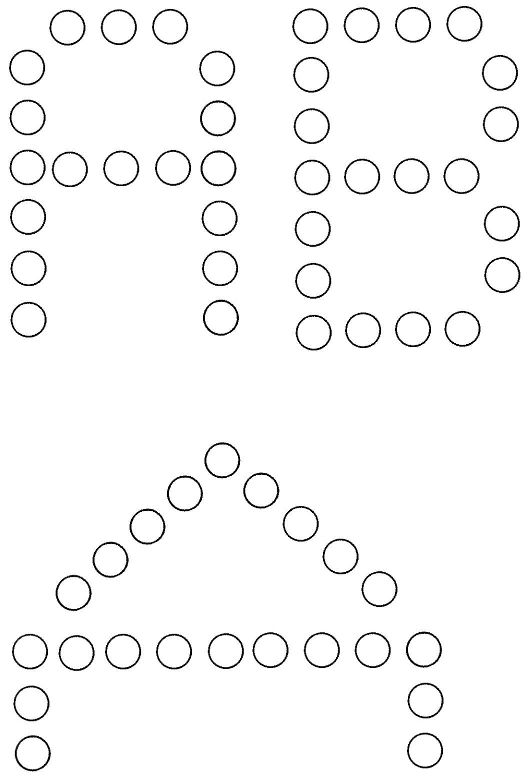
DOT-MATRIX PRINTING

If you look on the screen of your Radio Shack computer, you'll see that individual characters are formed by tiny dots, rather than smooth line segments. This type of character formation is called dot-matrix, because a matrix of dots is used to produce the character or symbol. Figure 2-1 shows how different characters are formed on several types of Radio Shack printers.

Dot-matrix printing is popular because it's relatively easy to make a print head composed of tiny wires that strike the paper through an inked ribbon to produce the character image. Most Radio Shack printers produce the character by printing one column of dots at a time. The microprocessor and program contained within the printer receives one character code, such as the value 65 for an "A", and then looks up the corresponding column configurations in a program table to find the proper columns to print, as shown in Figure 2-2. (The LPVII, DMP-100, and DMP-110 are dot-matrix printers, but use a somewhat different approach to forming the dot-matrix. See Figure 12-20.)

The dot-matrix print mechanism lends itself very nicely to graphics printing. The program in the printer simply bypasses the "table lookup" for the characters and prints whatever you've specified for the column instead, as shown in Figure 2-3.

Usually, only a portion of the character position is used to print the text character. A space is left above or below, or in both places, so that there is



DMP-100 (LP-VII) CHARACTER FORMATION 5 × 7 DOT MATRIX

DMP-110 CHARACTER FORMATION 9 × 7 DOT MATRIX

Figure 2-1. Dot Matrix Printing

white space between lines of characters. There's also space on one side of the character so that there's space between adjacent characters, as shown in Figure 2-4. When graphics printing is done, all of the vertical and horizontal dot positions can be filled so that there is no white space between dots, unless you want it. A dot-matrix printer printing in graphics, then, can fill the entire sheet of paper with dots, with no line- or character-spacing in between, as shown in Figure 2-5.

DAISY-WHEEL PRINTERS

Daisy-wheel printers are *impact* printers that print by striking a raised character against an inked ribbon. While a normal typewriter has only two characters per key lever, a daisy wheel contains all the characters in a character set for the printer. These characters are arranged like the petals of a daisy around a metal or plastic wheel.

The microprocessor in a daisy-wheel printer receives a character to be printed from the computer and then looks up the position of the proper petal for the character. It then rapidly rotates the daisy wheel to the proper position, and a hammer presses the petal against the inked ribbon, producing a character image, as shown in Figure 2-6.

The advantage of daisy-wheel printers is that high-quality type can be printed without dots, and different type styles can be produced by simply switching wheels. You can go from "Courier" to "Madeline" in a few seconds by switching wheels.

Printer Hint

DAISY-WHEEL QUALITY WITHOUT SWITCHING WHEELS

Wouldn't it be nice to have daisy-wheel quality without having to switch daisy-wheels? If you have a DMP-2100, you're almost there! The DMP-2100 is a dot-matrix printer with a special 24-wire print head, instead of the usual seven- or nine-wire print head. The 24 wires print in the same vertical space as a lower-density print head and produce a much more detailed print image. With such a fine resolution, it's possible to actually construct different type fonts, and that's what's done in the DMP-2100. Different type fonts can be selected in microseconds under software control, rather than the many seconds it takes to change daisy-wheels.

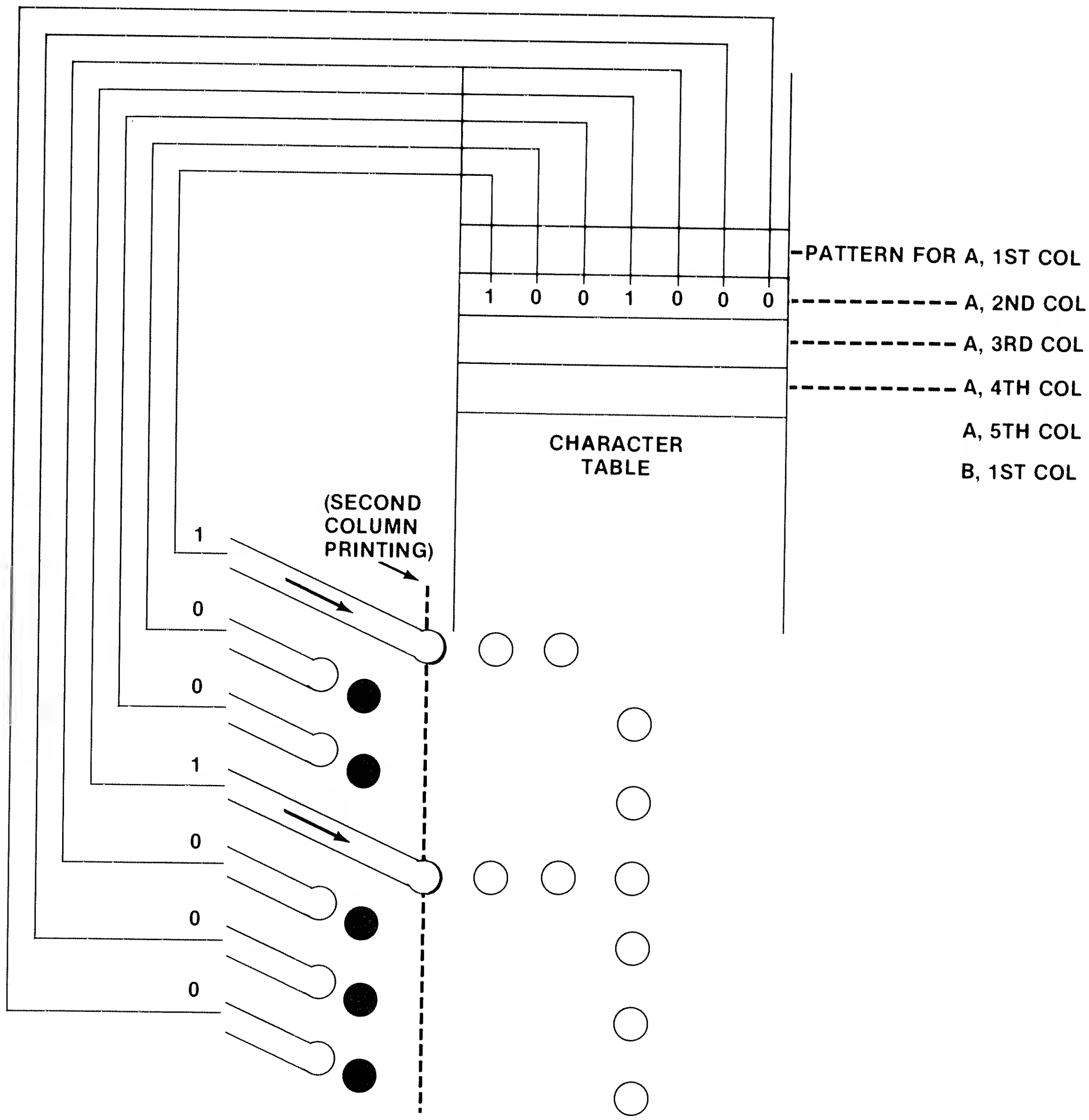


Figure 2-2. Column Printing

THERMAL PRINTERS

Thermal printers use a special aluminum-coated paper. The print mechanism is essentially a dot-matrix mechanism with print wire. Instead of the wires striking the paper through an inked ribbon, however, current flows through the wires causing a spark discharge to the paper, heating it, and turning it black.

Only the TP-10 printer uses this method currently, although the earlier Quick Printer I and Quick Printer II also were thermal printers.

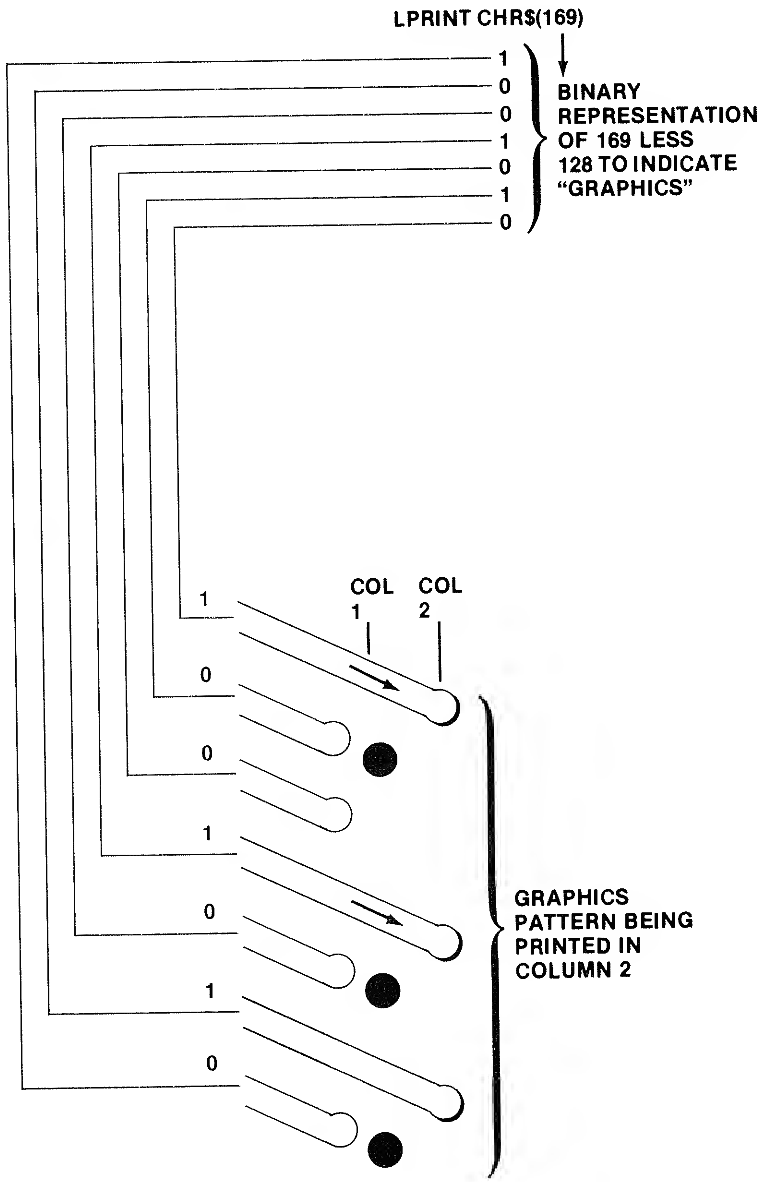


Figure 2-3. Graphics Column Printing

INK-JET PRINTERS

Ink-jet printers are also dot-matrix printers. In this case, however, the dots are formed by shooting droplets of ink onto the paper from a central reservoir. Because there's no ribbon, it's possible to draw from several ink supplies and produce colored printing, even to the extent of mixing colors to produce a spectrum of colors. This new type of printer has great potential because the dots are crisp and can be made very small. See Figure 2-7.

Ink-jet printers are a recent innovation in printers and Radio Shack has

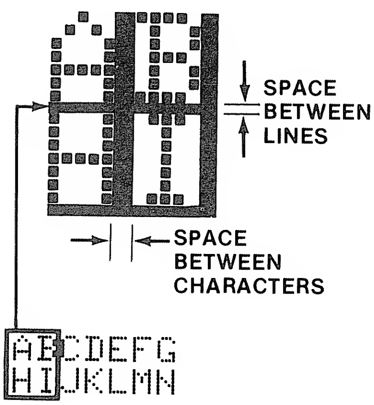


Figure 2-4. White Space in Text Printing

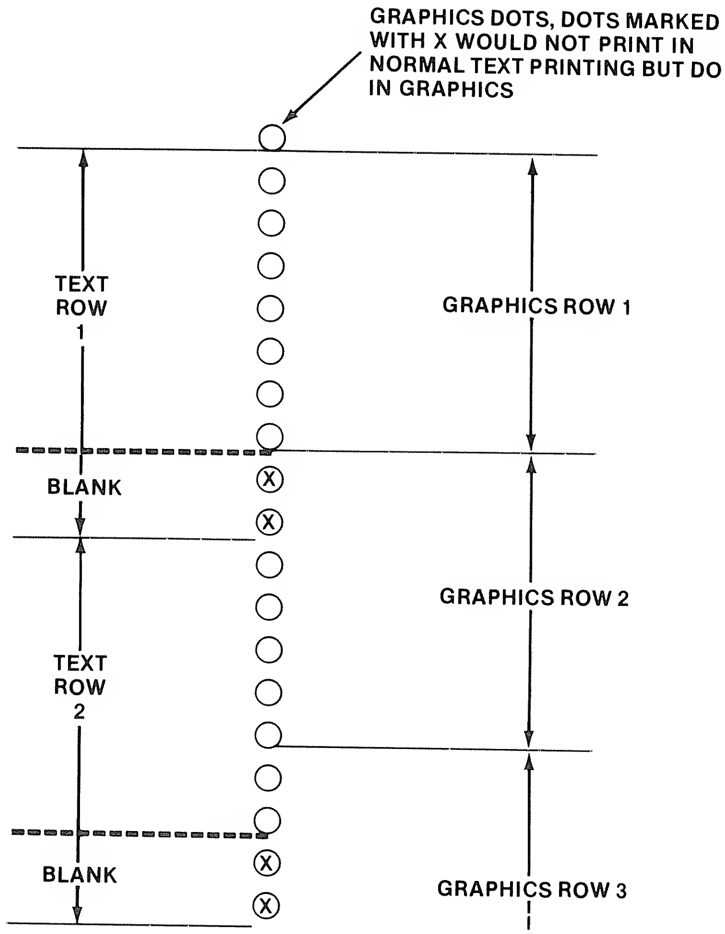


Figure 2-5. Graphics Print Density

one model currently, the CGP-220. It's capable of producing high-resolution color images.

PLOTTERS

Plotters are not really printers at all, although they can be used to produce text characters. A typical plotter has one or more pens and a mechanism to move the pens over a fixed sheet of paper or across a moving roll of paper, as shown in Figure 2-8.

The pen can be lifted from or put down on the paper under program control. It can also be moved to a new x/y coordinate as shown in Figure 2-9. A line segment is created by a series of tiny steps. If the step distance is small enough, smooth line segments and characters can easily be constructed.

The CGP-115 printer/plotter operates according to these general principles, and there are a number of other printer/plotters that work similarly. Since this is quite a different technique than the printing methods discussed here, and since the printer/plotters are not word processing oriented, we'll leave the details to another book.

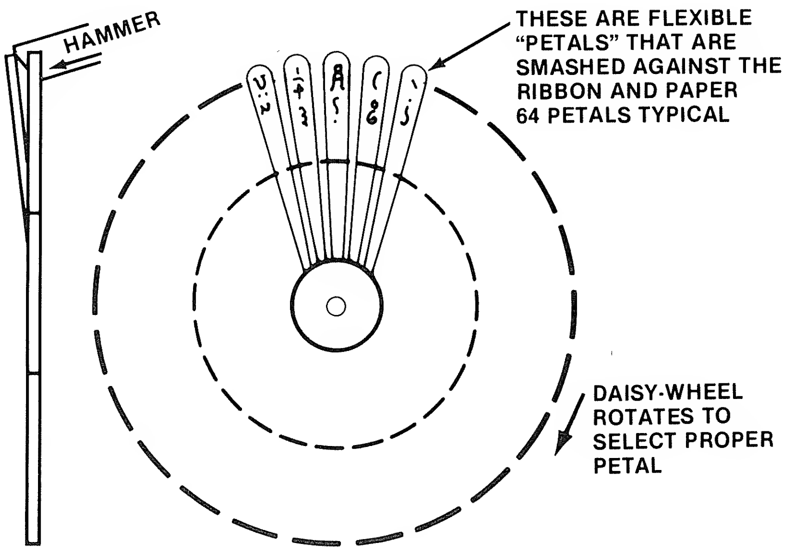


Figure 2-6. Daisy Wheel Printing

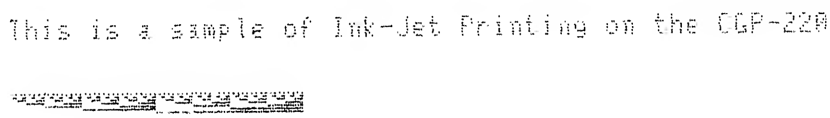


Figure 2-7. Ink-Jet Printing

TYPES OF PRINTING

Radio Shack has divided the types of printing that can be done with their printers into three categories:

Data Processing
Word Processing
Graphics Printing

The first two modes are very similar. The best way to describe the difference between data processing mode and word processing mode is to say that in word processing mode superscripting and subscripting of characters is possible, whereas in data processing mode it is not. In general, data processing mode is oriented toward getting a basic listing on the line printer, but in word processing mode the object is to make the printed text look "pretty." We'll look at the differences in detail in later chapters; for now just remember that the two modes are more alike than different. If you have an older printer, by the way, you probably won't have the two modes—you'll have only text

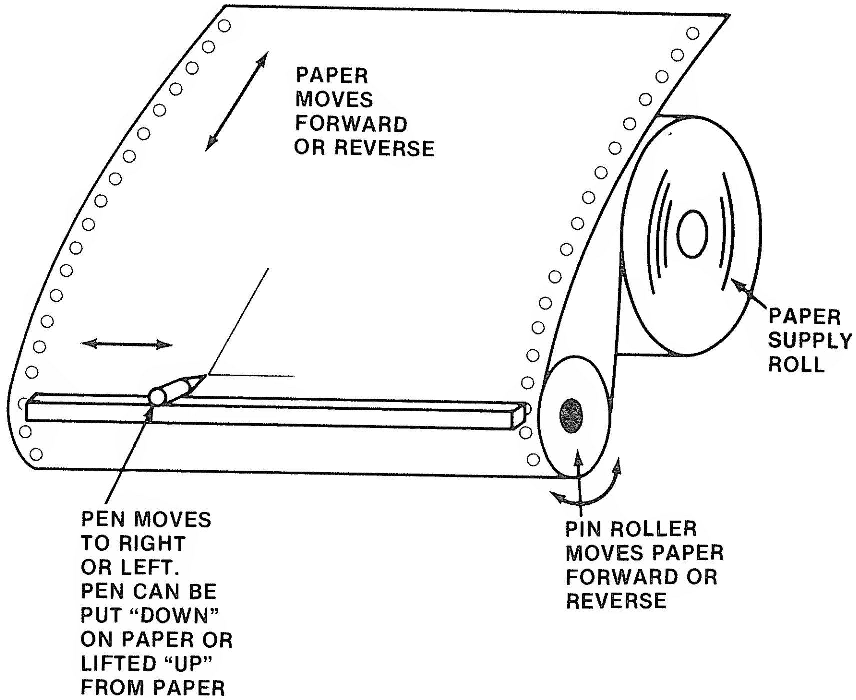


Figure 2-8. Plotter Operation

characters and graphics, or just text characters.

In the third category, the graphics mode, the printer output is a pattern of dots. It's usually impossible to print a preformed text character in this mode, and although it is possible to construct a pattern of dots that look like a text character, it's not an easy task. In this mode it's also possible to space over a single dot or point position at a time so that the graphics pattern can be printed anywhere along the line. This dot positioning is also possible in data or word processing mode in newer Radio Shack printers. A sheet of paper 8½ inches by 11 inches usually has on the order of 500 dot positions horizontally and 700 dot positions vertically, although this varies with the printer model. Any one of those dot positions can be printed or left blank to make up a "graphics picture" as shown in Figure 2-10.

The printer modes are set by sending a specific code to the printer from a BASIC program or another type of program. If the printer has more than one mode, it's usually very simple to switch from one mode to another under program control.

BASIC OPERATIONS IN DATA AND WORD PROCESSING

Paper

Figure 2-11 shows paper for Radio Shack printers. The paper comes in a few standard types. Most dot-matrix printers use fan-fold plain or green-bar paper. This paper is stacked in accordion folds and usually has 3000 or

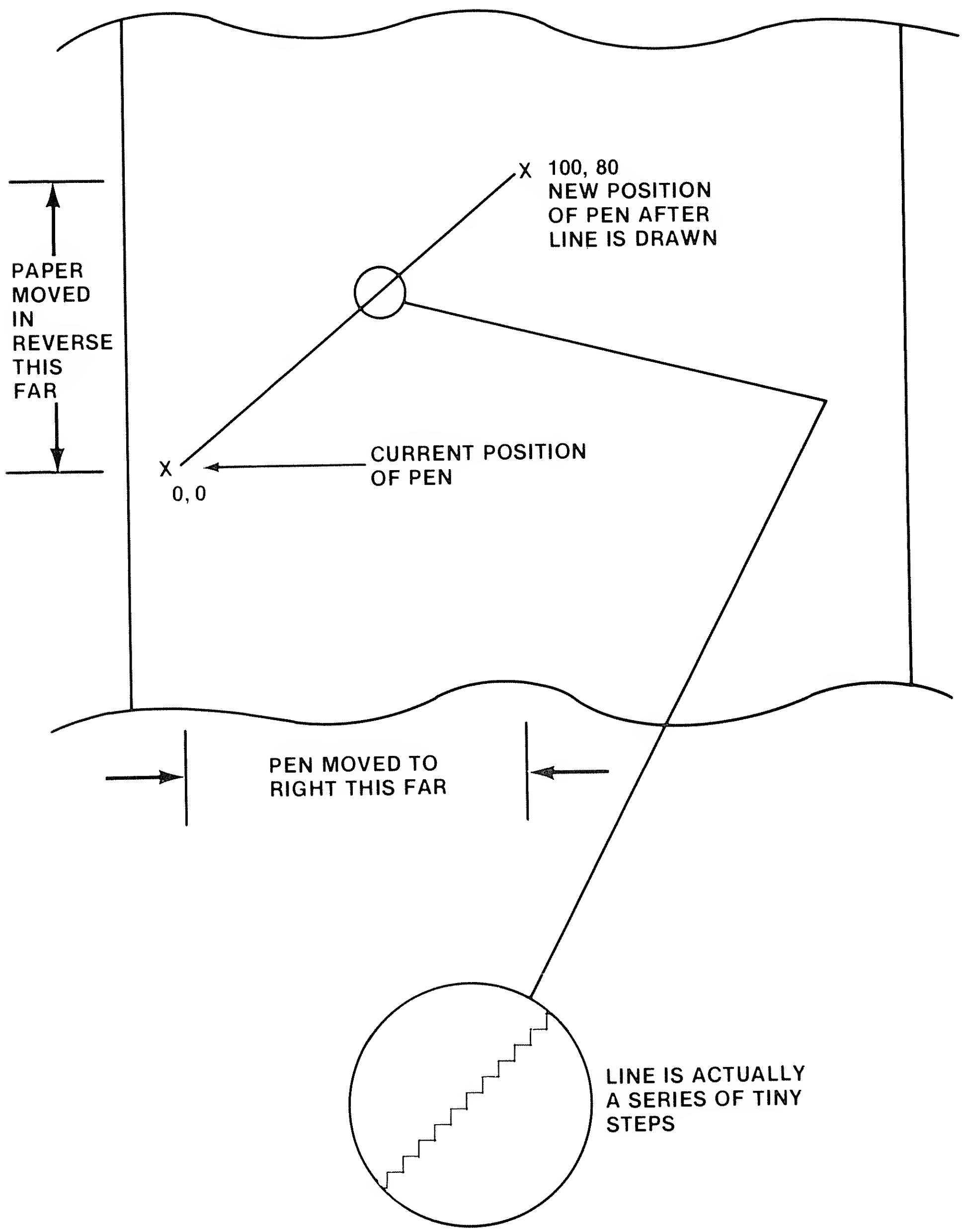


Figure 2-9. Line Segments in Plotters

5000 sheets in a 10- by 11- by 13-inch box. The paper has holes along its edge so that the feed-mechanism pins can move the paper up past the print mechanism. The most common paper width is 9½ inches with perforations along both sides so that the paper edges can be torn off. The result is a standard 8½-inch width. Of course, there are perforations horizontally every 11 inches so that the fan-fold sheets can be "burst' into standard 8½- by 11-inch sheets.

Some Radio Shack printers will only accept paper that is 9½ inches wide or less; other printers, such as the DMP-400 and DMP-2100 can accept 14-7/8 inch wide fan-fold paper.

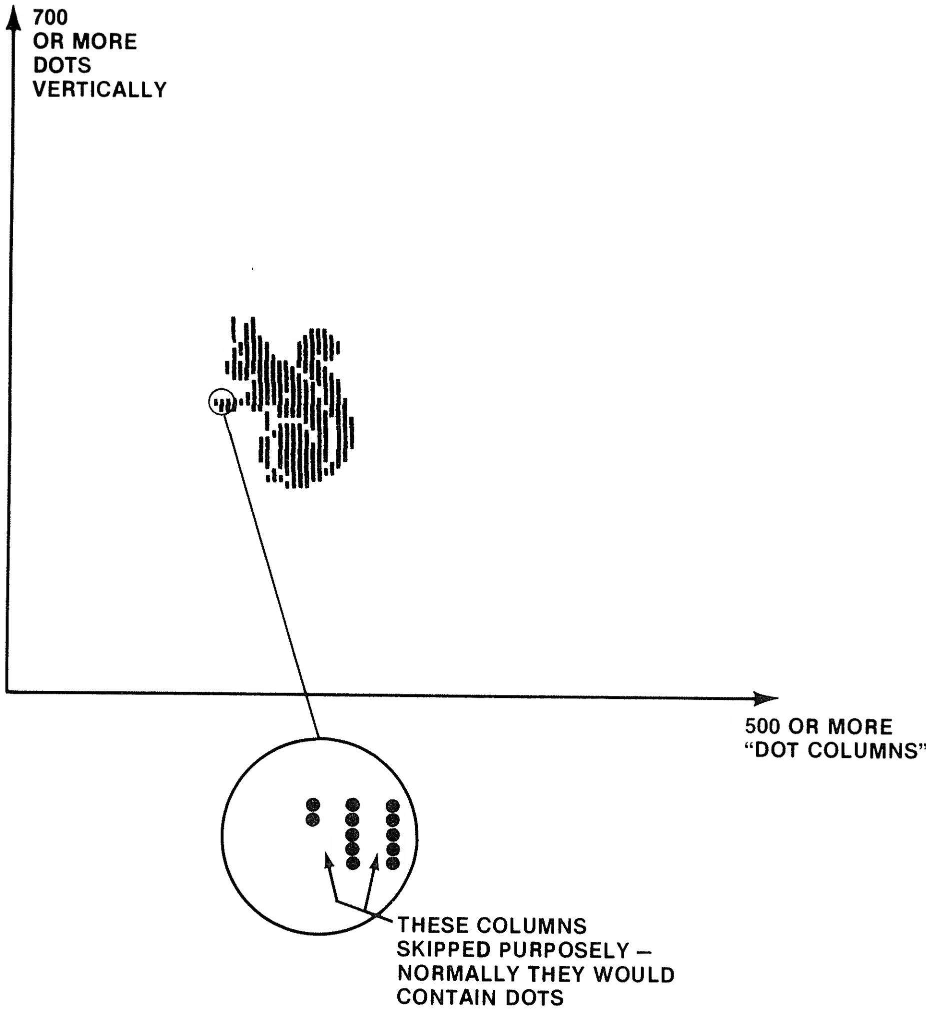


Figure 2-10. Graphics Printing

Printer Hint

MORE ON PAPER

The two stock paper sizes are 9½- by 11-inch fan-fold paper, and 14⅞- by 11-inch fan-fold paper. The first size has perforated edges so you can tear down the sheets to 8½- by 11-inch size. You can buy these two types of paper in a variety of colors and weights. The most popular smaller size is white in an 18- or 20-pound stock. The larger size comes in a "green-bar" style which, believe it or not, has green bars running horizontally across the paper.

In addition to basic printing stock, you can get a variety of forms for printing just about anything. If you print on three-inch by five-inch index cards, for example, you can buy tractor feed stock that tears down to three-inch by five-inch size. Similarly, you can get fan-fold envelopes, checks, invoice forms, letterheads, or mailing labels. Check with Radio Shack first and if they don't have it, there are literally dozens of computer forms suppliers in your area waiting to send you their catalogs.

Radio Shack thermal printers, and printers such as the CGP-115, use either thermal paper or paper that is a narrower width than the standard 9½ inch tear-down fan-fold paper. Usually this paper comes in rolls. Older model printers (LPI, LPII, etc.) also use roll paper.

In addition to the standard printing paper for printers, there's a host of labels, forms, envelopes, and other paper products that you can use on the printer. Most of these are oriented towards tractor-feed or pin-feed mechanisms, so your printer must be capable of feeding paper that has holes along each edge. In some cases, though, it's possible to use friction feed printers with the forms, although the printer may not space accurately over many pages.

PRINTER LINES

Figure 2-12 shows a typical 8½- by 11-inch printer page. (Chances are you have a printer that will accept that size paper.) Early printers (1978)

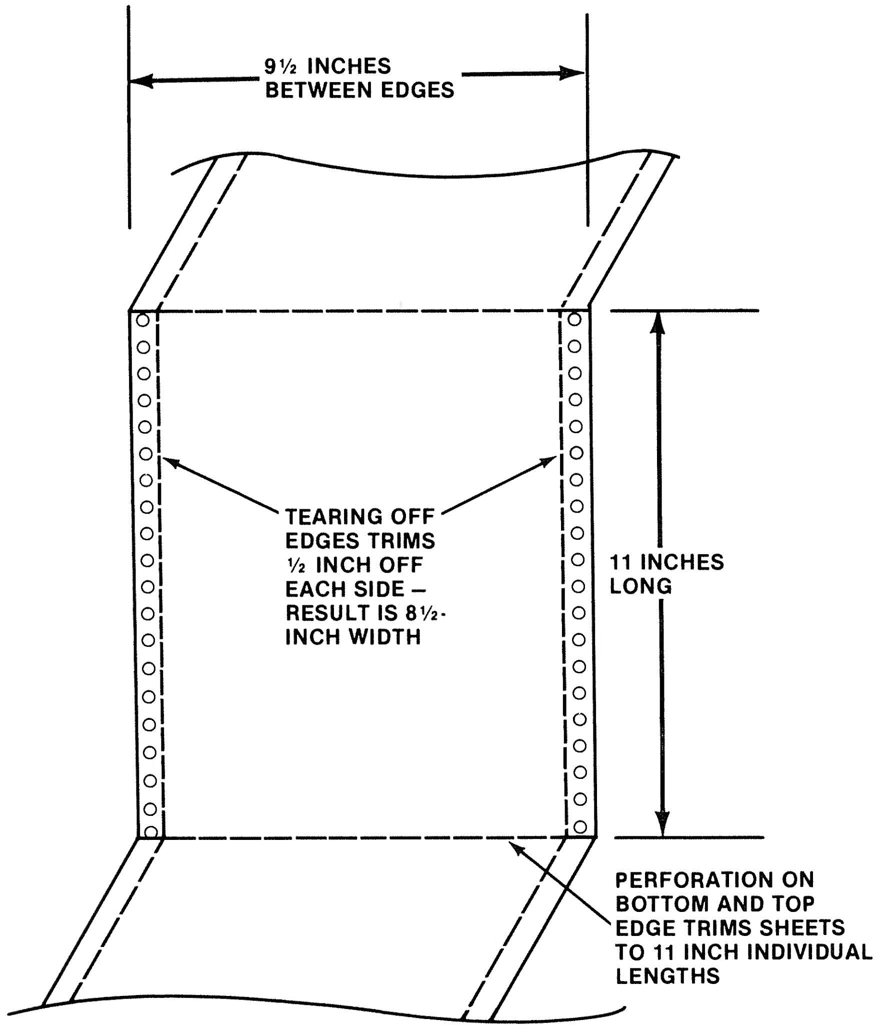


Figure 2-11. Printer Paper

printed text at ten characters per inch and six lines per inch. The maximum number of characters that can be printed horizontally for this type of spacing is 85, and the maximum number of lines per page is 66. This standard spacing is still in use today for most Radio Shack printers, except for printers using smaller roll paper such as the CGP-115 and TP-10.

Newer printers, though, can vary both the character spacing and the line spacing. You might want to vary the character spacing to print elongated or double-width characters to emphasize a heading. In other cases, you might want to print compressed characters to fit more text into a given space. There's also a mode in many printers called proportional spacing that makes text look nicer and also makes it easier to read. In proportional spacing, characters are given different spacing according to their width — an *i* is allocated less space than a *W*, for example. Figure 2-13 shows plain text at both ten characters per inch and proportionally spaced from a DMP-400.

The line spacing also affects the readability of documents. Many newer

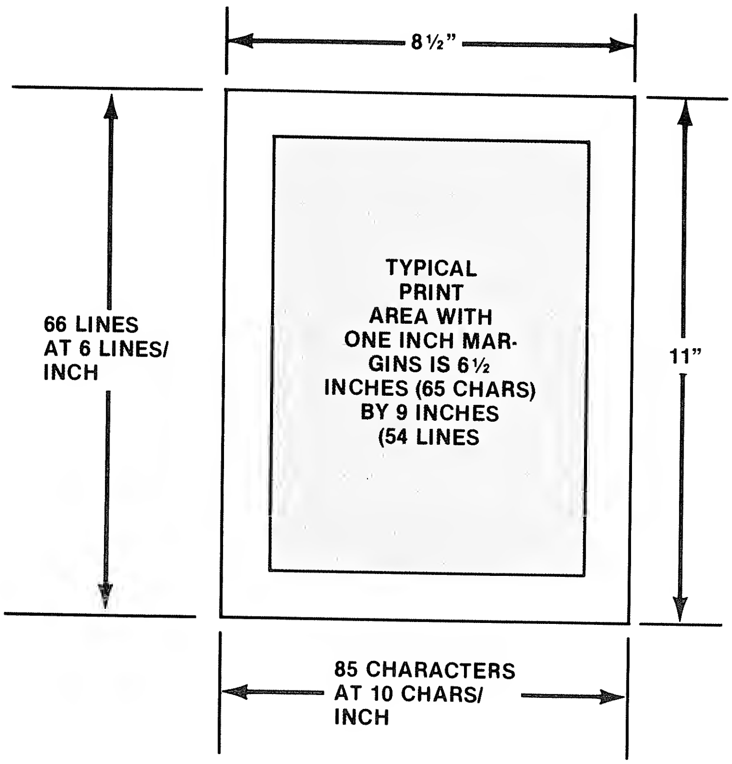


Figure 2-12. Typical Printer Page

'Here's the weather for the Orange County, California
area - warm with hazardous smog levels. We suggest
staying indoors and reading a good book on computer
printers.'

'Here's the weather for the Orange County, California
area - warm with hazardous smog levels. We suggest
staying indoors and reading a good book on computer
printers.'

Figure 2-13. Proportional Spacing

printers allow you to vary the line spacing, down to small increments. Figure 2-14 shows line spacing at the plain vanilla six lines per inch, eight lines per inch, and twelve lines per inch.

END OF LINE

In general, printers don't recognize the end of a line. If you have 9½-inch paper in your 14-7/8-inch carriage printer and print 100 characters at ten characters per inch without a new line, the printer will print past the edge of the paper and onto the platen, or roller. You (or the program you're running)

```
This shows a line spacing
of 6 lines per inch.

This shows a line spacing
of 8 lines per inch

This shows a line spacing
of 12 lines per inch.
```

Figure 2-14. Line Spacing

must know the maximum number of characters the paper will take. In almost all printers, though, if you attempt to print more than a given number of characters per line, the line will "wrap around" onto the next line, with the additional characters starting a new line.

LINE FEED AND CARRIAGE RETURNS

There are two confusing terms that describe how a new line starts under program control—line feed and carriage return. In early Radio Shack printers, sending a carriage return character to the printer caused the printer to position the print head to the beginning of a new line. This was a different action than in other (non-Radio Shack) printers, where a carriage return simply moved the carriage back to the beginning of the current line. In the non-Radio Shack printer, a line-feed character would then be sent to roll the paper up to the next line position. This action is shown in Figure 2-15.

Later Radio Shack printers, though, came with switch settings so that the printer would do either a carriage return, line feed or simply a carriage return when it received a carriage-return character. Radio Shack software generally sends only a carriage-return character for a new line.

A *carriage return/line feed* or *carriage return only*, then, results in the printer going to the next line. In older printers, this means that the printer will advance 1/6th inch for the new line; in more recent printers, the printer will advance 1/6th inch, 1/8th inch, 1/12th inch, or whatever it has been programmed to do. We'll show you how to do this in a later chapter.

MOVING THE PAPER

A *top-of-form* operation results in the paper being ejected one full page. If the line spacing is at six lines per inch, this operation will be the equivalent of 66 line feeds. Top-of-form operations position the paper to the first printing position on the next page.

Early Radio Shack printers could only move the paper forward to the next line for printing. However, more recent printers can also move the paper backwards. This allows you to do such things as plotting graphs without having to first construct a "print image" of the way the graph should look, and then print it out from top to bottom. Not every Radio Shack printer has this option, however.

Printer Hint

WHY HAVE A CARRIAGE RETURN AND LINE FEED?

Here again, the reason for having both a carriage return and line feed is largely historical. Early teletypewriters were slow devices—on the order of five or six characters per second. It took time for the carriage to return from the end of a line back to the beginning, and so it was advantageous to be able to space to the next line without waiting for the carriage return. (This is perfect for putting things in columns, for example, a line feed, a few backspaces, and you're ready to print again.) With intelligence in printers, the print position can be found easily without processing a separate line feed and carriage return, but early Radio Shack printers were perhaps a little too smart in doing both a carriage return and line feed.

GRAPHICS OPERATIONS

In one sense, graphics operations are a lot more simple than text operations. There are only a few operations that can be done in this mode: print-

Examples:

Case 1

This is the end of the line [CR] [LF]
And this is the next line.

Case 2

This is the end of the line [CR]
And this is the next line.

Case 3

This is the end of the line [LF]
And this is the next line.

How the Examples Print on Non-Radio Shack Systems:

Case 1

This is the end of the line
And this is the next line. (Normal)

Case 2

And this is the next line (Overprinted)

Case 3

This is the end of the line
                                        And this is the next line.
                                        (No carriage return)

How the Examples Print on Radio Shack Systems:

Case 1

This is the end of the line

And this is the next line. (Two line feeds)

Case 2

This is the end of the line
And this is the next line. (Normal)

Case 3

This is the end of the line
And this is the next line. ([LF] causes [CR], [LF])

[LF] = Line Feed Character 10
[CR] = Carriage Return Character 13

Figure 2-15. Line Feed Vs. Carriage Return

ing a graphics column, doing a carriage return/line feed, and positioning the print head to a new dot position along the line.

The basic unit of location with a graphics printer is a *dot column*. Generally, this is an offshoot of the size of the dot matrix used in defining a character for dot-matrix printers. If characters are twelve dots wide, for example, there will probably be 960 dot positions across the line for every 80 characters (see Figure 2-16). Most Radio Shack printers allow you to position the print head (via a special set of codes) to any one dot or point position. Line feeds are used to position the print head vertically.

Suppose we wanted to print the graphics pattern shown in Figure 2-17. One graphics command would position the print head to the upper left-hand column. Another graphics command would print the column. Two more graphics print commands would print the two remaining columns along the first line. After the last column was printed, a line feed would move the print head to the next graphics line (not necessarily the same as a text line, depending upon the printer). Another position command would move the print head to the extreme left position for the second graphics row. Six more graphics print commands would print the second row. The sequence would then be repeated one more time for the last row.

The sequence of operations above is typical for every Radio Shack graphics printer. There is a great deal more work in defining the graphics picture and programming the sequence of commands than in doing the actual printing.

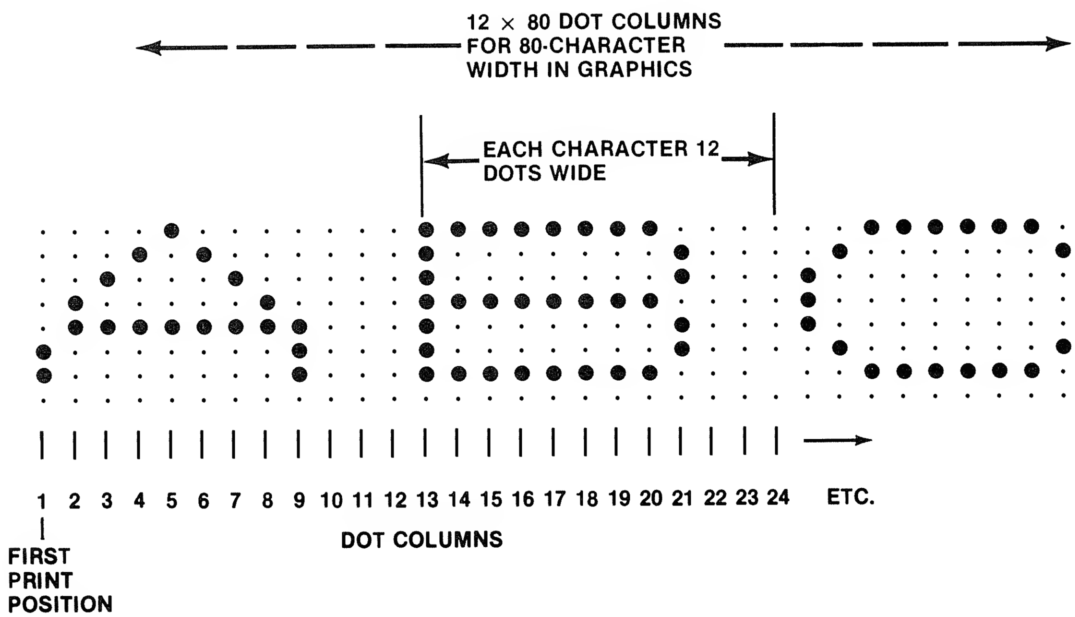


Figure 2-16. Dot Column Positions

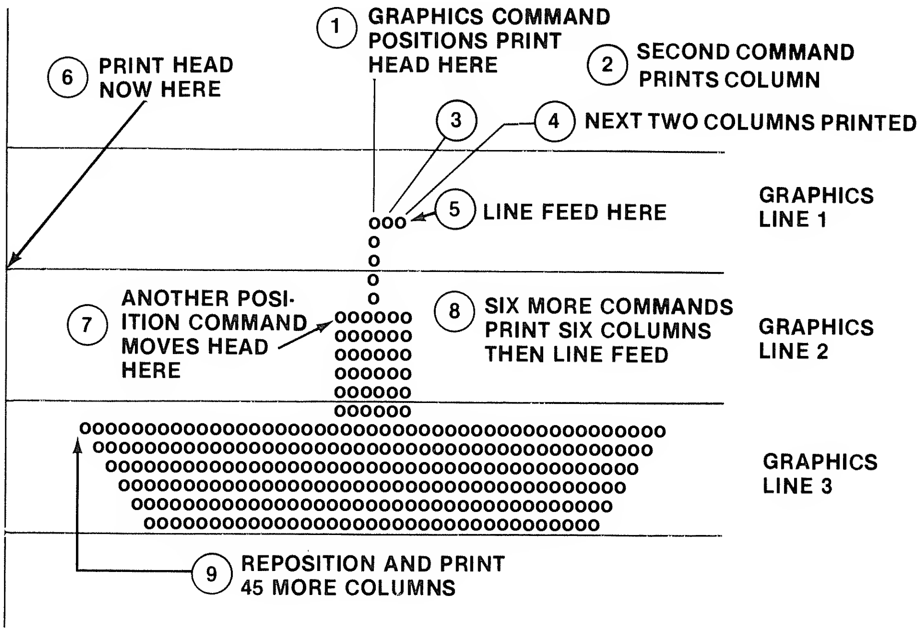


Figure 2-17. Graphics Pattern Positioning

Printer Hint

A GRAPHICS PICTURE IS WORTH A THOUSAND HOURS

As mentioned in an earlier Hint, it takes a long time to process graphics data. We didn't mean to be too pessimistic in mentioning 25 hours, however. A typical full-page graphics picture might consist of 1/4 of the page area devoted to graphics—about 20 square inches. Assuming that each graphics element is about 1/100 of an inch by 1/10 of an inch, it would take about 20,000 elements to make up a graphics picture. Assuming that each element could be processed in BASIC at the rate of about 100 elements per second, it would take about 200 seconds to print the picture, an acceptable rate. However, the 100-element-per-second rate is the best case and the processing time can be 10 or 20 times slower for heavy "number crunching" of graphics data.

CHAPTER 3

HOW YOUR PRINTER COMMUNICATES WITH YOUR SYSTEM

In this chapter we'll look at how your printer is connected to your Radio Shack computer and how the two devices "talk" to one another. There are several different levels of this communication, and we'll discuss what's involved in printer communication on each level.

PARALLEL VS. SERIAL CONNECTIONS

There are two ways that a Radio Shack printer can be connected to a Radio Shack computer system, or to another manufacturer's system—*parallel* and *serial*. Some Radio Shack printers are only serial devices, some are only parallel devices, and some are both parallel and serial.

PARALLEL CONNECTIONS

A parallel printer is connected to a computer system as shown in Figure 3-1. Data is transferred one *byte* at a time and the actual transfer takes a few millionths of a second. The parallel connection uses eight lines to transfer each of the eight bits that make up one byte of data. By the way, generally, one character of text (including spaces) is held in one byte of data. To print the text "FORT WORTH TORNADO SEASON," 25 bytes of data are transferred to the printer. Each time a byte is transferred, the eight bits making up the character are transferred over the eight lines simultaneously. In addition to the eight data lines there are lines for other signals ("out of paper" status, "ready" status, etc.), so the resulting cable is about 14 separate lines. (Radio Shack parallel printer cables generally use 34 lines but many of these lines are ground signals.)

SERIAL CONNECTION

A serial printer is connected to a computer system as shown in Figure 3-2. Data is still transferred one byte at a time for each character, but that byte

Printer Hint

MORE ON PARALLEL CONNECTIONS

The parallel connection is also called a Centronics port, after the printer manufacturer who developed it. It uses a 36-pin Centronics connector with a standard configuration, and is the same connector used on non-Radio Shack printers and computer systems. The parallel plug on the Radio Shack computer end is typically an edge connector, or "header," but the electronics signals all conform to the Centronics specification—you just have to know how to connect the Radio Shack pins to the 36-pin Centronics connector. (Another type of plug used on IBM PC-type systems like the Tandy 2000 and 1200 is the DB-25, a 25-pin connector.)

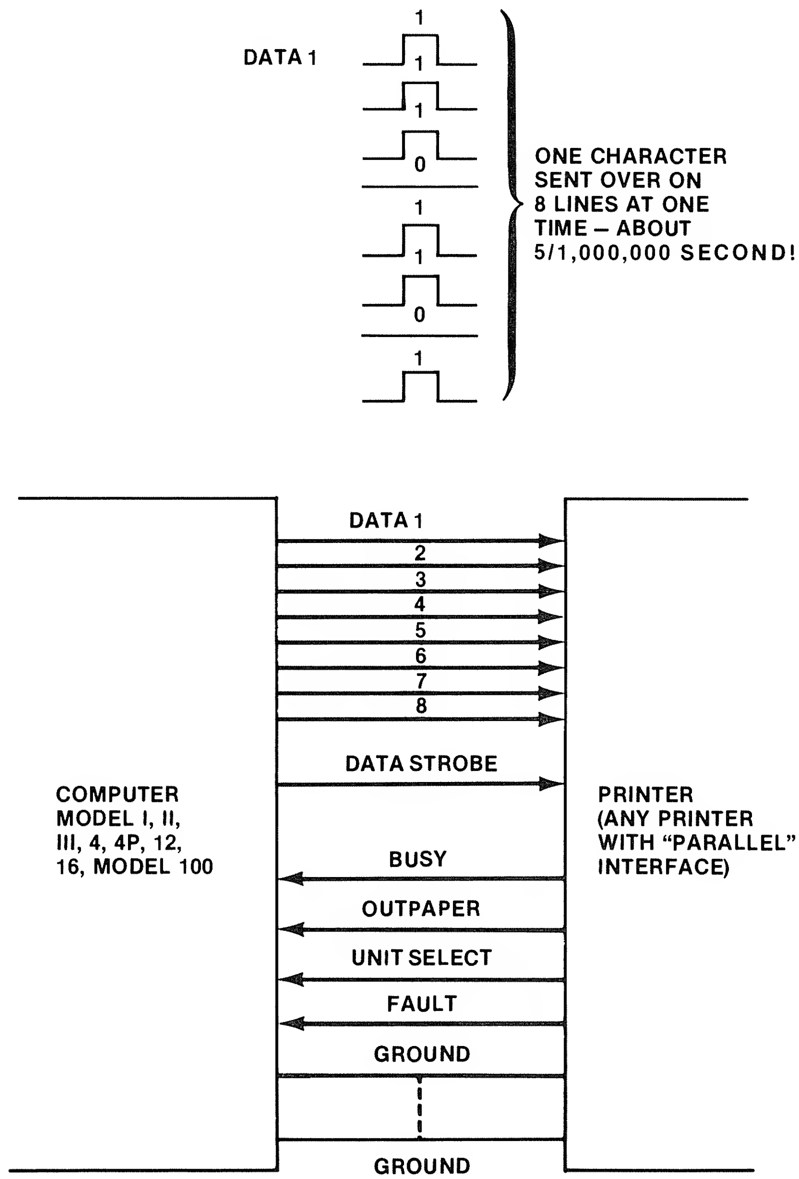


Figure 3-1. Parallel Printer Connection

is split up into eight bits and the bits are sent over a single pair of wires during a specific time period.

The process is very similar to signalling over a long distance with a buzzer and remote switch. You could prearrange with a friend that you would send either a buzz or no buzz, representing either a binary 1 or 0, at about noon. One second after a "signalling buzz," your friend would look for the first data buzz, two seconds after for the second data buzz , and so forth, for a total time of eight seconds, as shown in Figure 3-3. Your friend would record either a buzz (a one) or no buzz (a zero) at precise one-second intervals after the first buzz, and after eight seconds he'd have the eight bits of data making up the byte, or character.

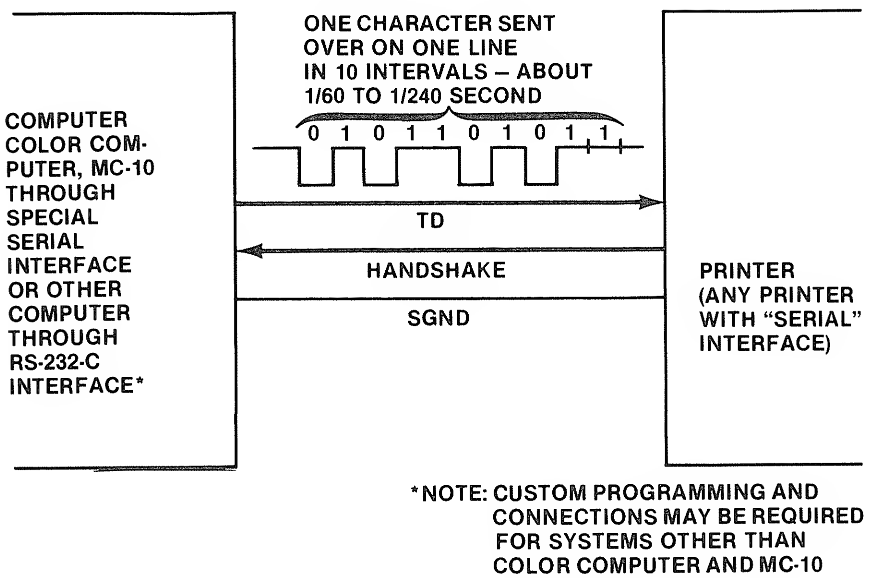


Figure 3-2. Serial Printer Connection

Of course, when a computer sends data it is much faster and can send the eight bits in 1/30 to 1/240 of a second, or faster.

WHICH IS BEST?

The advantage of a parallel connection is that it is faster and may not require a special *serial driver* program (which we'll discuss later). The advantage of a serial connection is that the cable is less expensive, as it requires fewer wires (four on the Color Computer and MC-10), and the length of the cable can be a great deal longer than the parallel cable.

You can easily tell whether your printer uses serial, parallel, or both. First, of course, look in your manual; it should tell you what type of connection is required. If you're one of those people who hate manuals (aren't we all?), look on the back of your printer. If the printer has a parallel connection, it will look like Figure 3-4, a 36-pin *Centronics* connector. If the printer has a serial connection, it will look like Figure 3-5, a four-pin *DIN* connector.

Table 3-1 shows the type of connection for Radio Shack printers available at this time. There will be additional printers out by the time this book is published that are not in the table; chances are most of these will be both parallel and serial.

If your printer is both parallel and serial, you may have to switch between one and the other with a switch on the back of the printer, as shown in Figure 3-6. In some printers, switching may be done automatically when you plug in one connection or the other.

SETTING THE SWITCHES

Most new printers have a set of switches called *DIP* switches which must be set before the printer is powered on. The switches look like Figure 3-6, and are usually located on the rear of the printer (they're inside some of the printers).

Printer Hint

MORE ON SERIAL CONNECTIONS

Radio Shack printers with serial connections conform loosely to the RS-232-C serial communications specification. This spec defines the way data is sent through as few as two wires for transmission, in one direction, or three wires for transmission in two directions. Radio Shack serial connections have four wires—one for incoming (to the printer) data, one for outgoing data, a ground wire, and a status line. Standard Radio Shack transmission rates are 600 or 1200 bits per second, corresponding to 60 or 120 characters per second.

The normal RS-232-C connector is a 25-pin DB-25 connector. This connector is used on all Radio Shack computers except for the Color Computer and MC-10. Serial printer cables use only four of the 25 RS-232-C signals. Most of the remaining signals are used for telephone modem connections.

Note: There are enough differences between the RS-232-C specification and the Radio Shack implementation of the spec that connecting a Radio Shack printer to a non-Radio Shack computer may require special programming and connections.

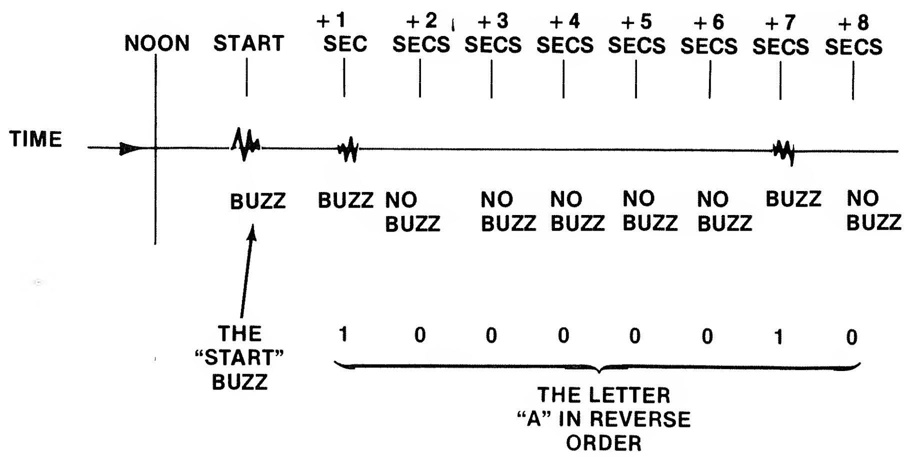


Figure 3-3. Serial Transmission

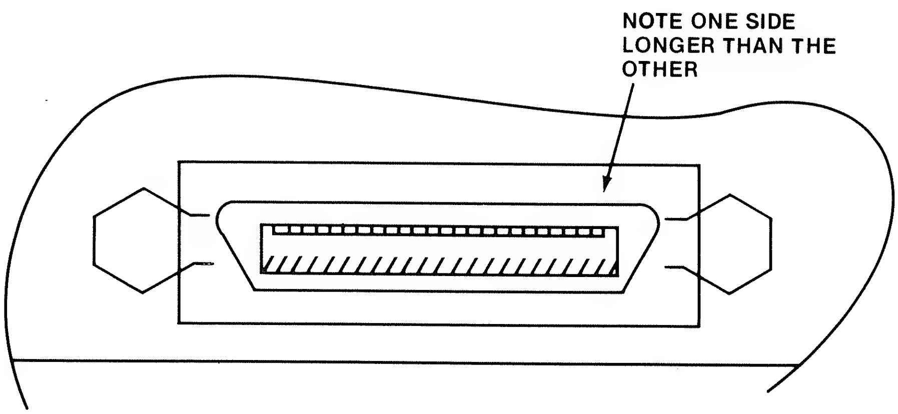


Figure 3-4. Parallel Connector (Typical)

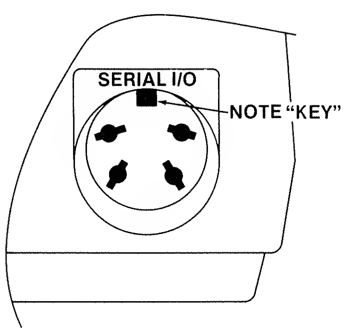


Figure 3-5. Serial Connector (Typical)

The most important point to remember about the switches is this: ON MOST PRINTERS THE SWITCHES ARE READ ONLY WHEN THE PRINTER IS FIRST TURNED ON. IF YOU CHANGE THE SWITCH SETTINGS, THEY WON'T BE READ AGAIN UNTIL THE PRINTER IS TURNED OFF AND THEN ON! (If I only had a new ribbon for every time I've been caught on this important point, all my listings would be dark . . .)

The number and types of switches vary, but we'll try to give you a description of most of the functions. The DIP switches for all printers are shown in Figure 3-7.

SERIAL VS. PARALLEL

This switch selects either a serial or parallel connection as described above. It won't be found on printers that are only parallel (LPI through LPVI, daisy-wheel printers DW-I and DW-II, DMP-2100), or on printers that are only serial (TP-10).

Table 3-1. Parallel and serial connections for Radio Shack printers

LPI:	Parallel only
LPII:	Parallel only
LPIII:	Parallel only
LPIV:	Parallel only
LPV:	Parallel only
LPVI:	Parallel only
LPVII:	Parallel or 600 BPS Serial
LPVIII:	Parallel or 600/1200 BPS Serial
DMP-100:	Parallel or 600/1200 BPS Serial
DMP-110:	Parallel or 600/1200 BPS Serial
DMP-120:	Parallel or 600/1200 BPS Serial
DMP-200:	Parallel or 600/1200 BPS Serial
DMP-400:	Parallel or 600/1200 BPS Serial
DMP-420:	Parallel or 600/1200 BPS Serial
DMP-500:	Parallel only
DMP-2100:	Parallel only
DW-I:	Parallel only
DW-II	Parallel only
DW-IIB:	Parallel only
DWP-210:	Parallel or 600/1200 BPS Serial
DWP-410:	Parallel only
CGP-115:	Parallel or 600/1200 BPS Serial
CGP-220:	Parallel or 600/2400 BPS Serial
QP-I:	Parallel only
QP-II:	Parallel or 600 BPS Serial
Plotter/Printer:	Parallel only

When there's a serial/parallel switch, there's also a switch to set the *bits per second*, or *baud rate*. You have a choice between 600 bits-per-second (60 characters per second) or 1200 (120 characters per second). The CGP-220 selection is between 600 and 2400 bits per second. If you're using a Color Computer or MC-10, use the 600 BPS position, because that's the standard baud rate for the MC-10 and Color Computer.

On the LPVII and LPVIII, you'll have a choice of "7BS" or "8BS." Select 7BS for 7-bit serial, and we'll describe how to use 8-bit serial later.

CR VS. CRLF

On the newer printers there's a switch for NL (new line) or CR (carriage return). On older printers there may be a switch for CRLF or CR. This is the carriage return/line feed problem described in the previous chapter. Setting the switch to NL or CRLF gives a carriage return *and* a line feed whenever a carriage-return character is received by the printer. This is the normal setting for Radio Shack printers. Setting the switch to the CR position results in just a carriage return being done when a carriage-return character is received by the printer. This position should be used when a Radio Shack printer is being used with another computer system that uses this convention, such as the Tandy 1200 or Tandy 2000.

In newer printers there's another switch marked similarly— LF and NL. This switch controls the action of the line-feed character. As we discussed in the previous chapter, a line feed was originally used to cause a new line action. The standard Radio Shack action is to do only a line feed without a carriage return, so set this to LF.

TANDY VS. ASCII CHARACTERS

Another switch on newer printers enables you to select either Tandy/ Modified ASCII or ASCII characters. We'll get into ASCII codes in another chapter, but for now, let's just say that the standard setting for this switch is "TANDY" or "Modified ASCII."

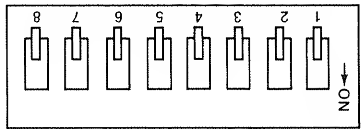


Figure 3-6. Dip Switches on Printers (Typical)

Printer Hint

CHANGING THE COLOR COMPUTER PRINTER PARAMETERS

The Color Computer communicates with the printer by a software printer driver. The printer driver uses values in several RAM memory locations to define the baud rate, line delay (delay on carriage returns for certain printers), comma field width (tab width), last comma field (last tab field), line printer width (number of characters per line) and character count (number of characters in current line). You can read or write these parameters at any time by using a BASIC PEEK (read) or POKE (write) command. For example, to change the baud rate to 1200, you'd enter this command:

```
100 POKE 95,0: POKE 96,41
```

Here are the locations:

Location (Decimal)	*Description*
149	Baud rate. Use 0 for 300, 600, 1200, or 2400.
150	Baud rate. Use 180 for 300, 87 for 600, 41 for 1200, 18 for 2400.
151	Line delay. Use 223 for each second delay.
152	Line delay. Use 0.
153	Use 16 for normal tabs.
154	Use 112 for last tab field.
155	Use 132 or whatever line length you'd like.
156	Normally, you wouldn't alter this; it's just used to hold the current character count for the line.

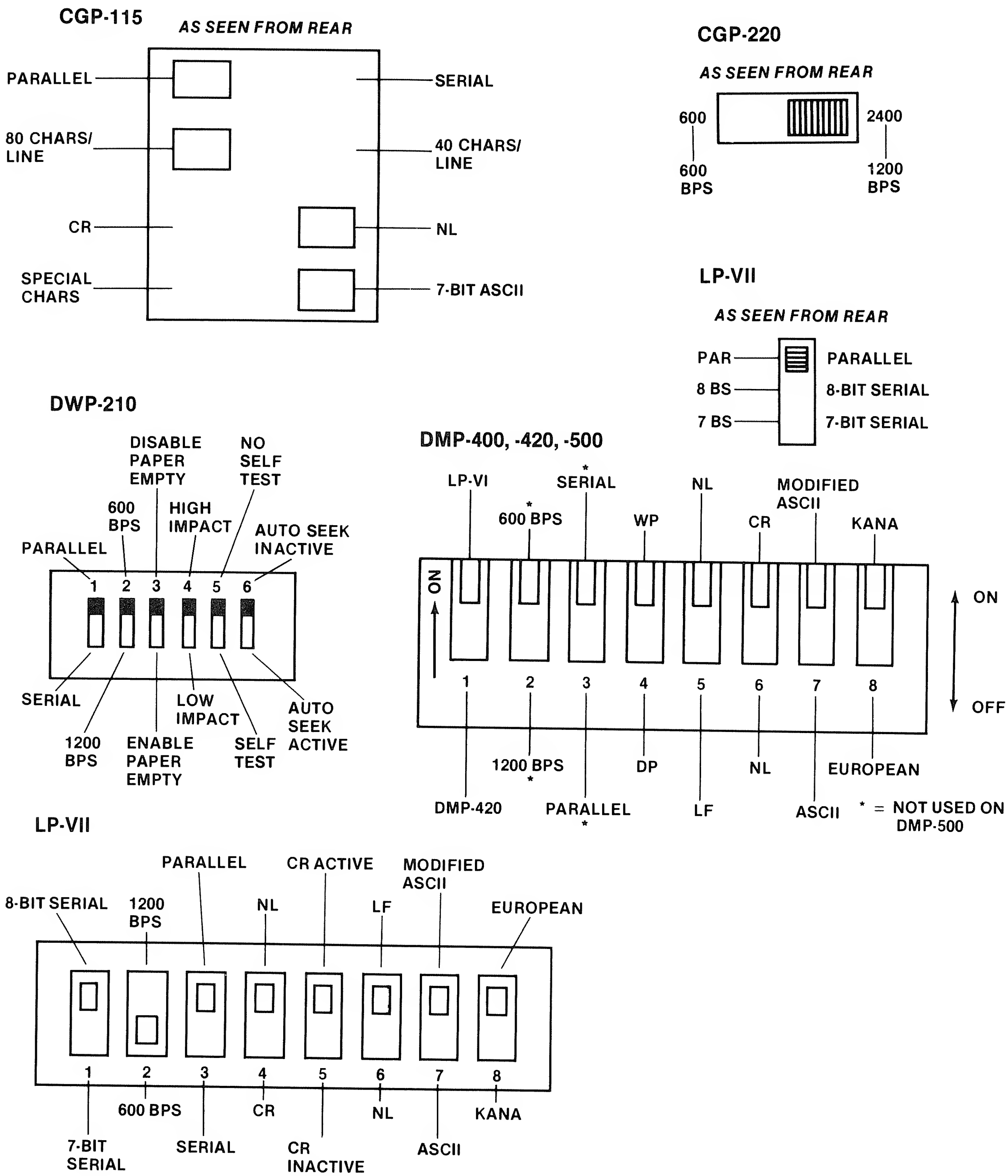
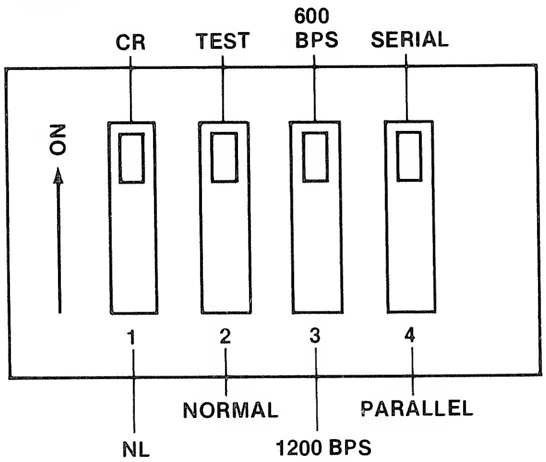
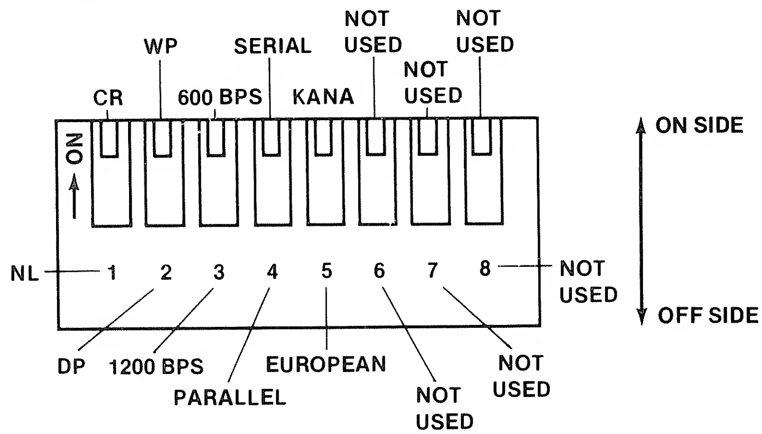


Figure 3-7. Dip Switch Settings

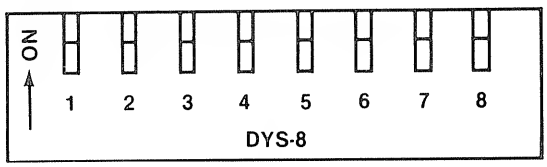
DMP-120



DMP-200

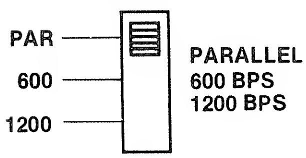


DMP-2100



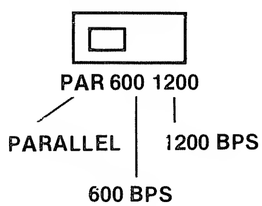
DMP-100

AS SEEN FROM REAR



DMP-110

AS SEEN FROM REAR



	OFF	ON
8		
7	ASCII CHARACTER	TANDY CHARACTER
6	CARRIAGE RETURN WITH LINE FEED	CARRIAGE RETURN ONLY
5	LINE FEED ONLY	LINE FEED WITH CARRIAGE RETURN
4	DP MODE	WP MODE

3	2	1	CHARACTER
OFF	OFF	OFF	STANDARD-10
OFF	OFF	ON	STANDARD-12
OFF	ON	OFF	CONDENSED-16.7
OFF	ON	ON	STANDARD-10
ON	OFF	OFF	CORRESPONDENCE-10
ON	OFF	ON	CORRESPONDENCE-12
ON	ON	OFF	STANDARD-10
ON	ON	ON	PROPORTIONAL

KANA VS. EUROPEAN

Another switch on newer printers selects either Japanese Kana characters or European characters. This is not what you'd think. It doesn't affect the normal text you'll send to the printer, but will affect a *subset* of characters that are selected by special codes. Newer printers have the ability to print the Japanese characters, which are phonetic representations, or special European characters such as the German umlaut or British pound sign.

DP VS. WP

Another switch lets you select either data processing or word processing actions when you power up. Remember that the switches are read only on power up—selecting one or the other will automatically result in those actions after the printer is turned on. However, you can change this selection under program control, so you aren't forced to adhere to the switch setting. In word processing mode, superscripts and subscripts may be done, along with some other functions we'll cover later. The normal setting for this is WP.

NEW VS. OLDER LINE PRINTERS

The DMP-400, -420, and -500 were upgrades of the earlier LPVI and V. The newer printers operate a little differently, compared to the older models. To make the upgrade as smooth as possible, the newer printers have switch settings to make the printers act exactly like the older models. Unless you're running old software, use the settings for the DMP-400, -420, and -500.

MISCELLANEOUS SWITCH SETTINGS

The CGP-115 has a switch setting for printing 40 or 80 characters per line. Use whichever you prefer, although the 80-character mode is more difficult to read. The DMP-2100 has three switches which select the typeface to be used after power up. Refer to the manual for these options.

THE BASIC CHARACTER FLOW

Whether your printer uses parallel or serial connections, the basic process for communicating with the printer is the same, and is shown in Figure 3-8. The characters are sent out, one at a time, by the program in your computer. This program may be an applications program, such as VisiCalc, a COBOL program, a BASIC program, an assembly language program, or another type of program. For each type of program the action is the same—we'll illustrate these actions with a short BASIC line:

```
100 LPRINT "THE QUICK PROGRAMMER JUMPED OVER THE
     LAZY ENGINEER"
```

When the BASIC *interpreter* program processes this message, it first recognizes the "LPRINT" (PRINT#-2, in the Color Computer) as a command that causes a message to be printed. It then processes the remainder of the line and finds that it is required to print "THE QUICK PROGRAMMER JUMPED OVER THE LAZY ENGINEER". The double quotes around the message are used to tell the BASIC interpreter where the message begins and ends. They are not printed.

The BASIC interpreter then sends each character of the message to the printer, starting with the *T* in "THE". This involves an assembly language program in the interpreter called the "printer driver," which we won't discuss. Suffice it to say that the 50 characters in the message are sent to

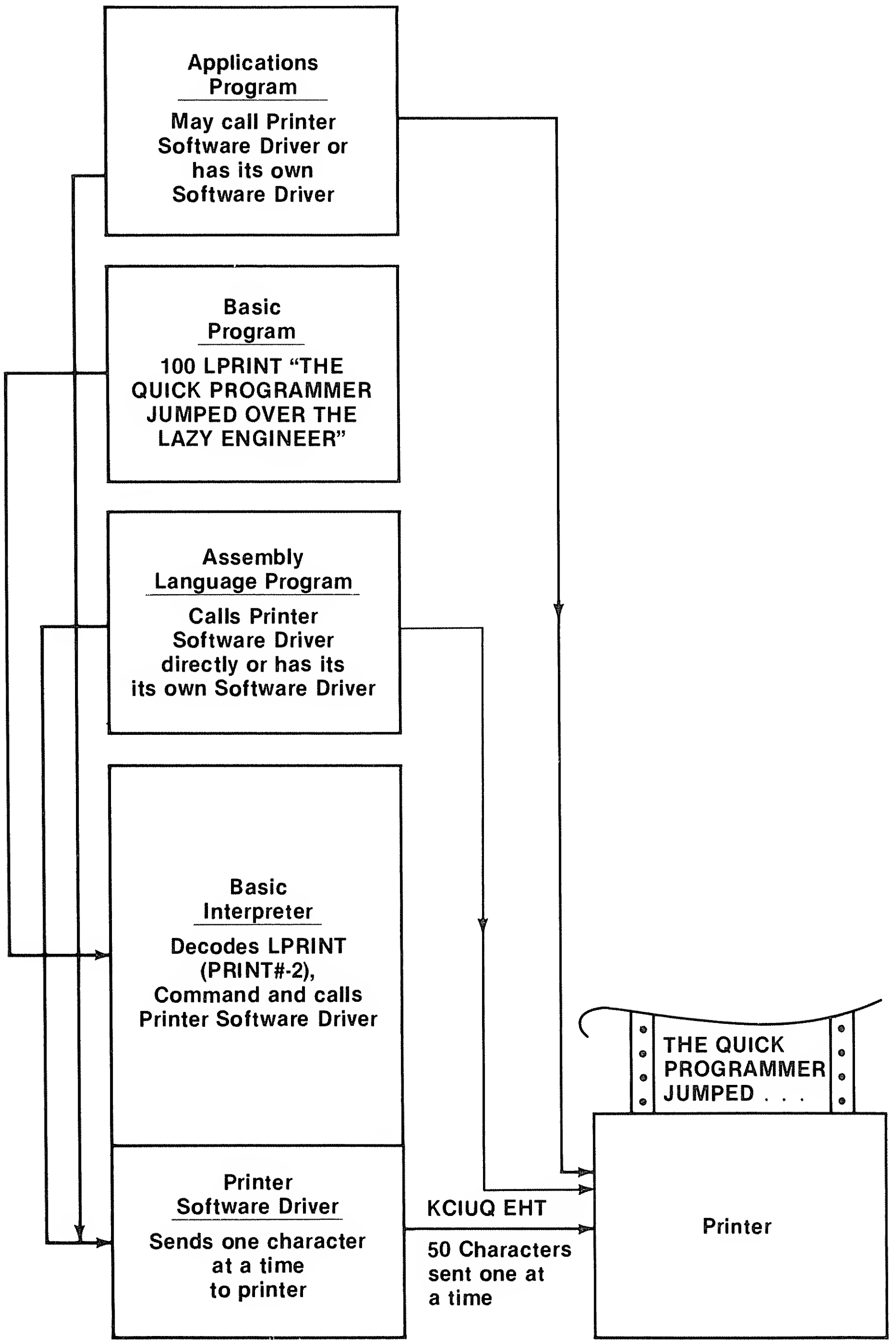


Figure 3-8. Printer Communication

Printer Hint

ASSEMBLY LANGUAGE PRINTER DRIVER

The printer driver in the BASIC interpreter is an assembly language program that communicates with the printer hardware. A typical driver consists of several hundred bytes of machine-language code. It takes an input string of characters to be printed (all found at one location in RAM) and strips off the characters one at a time for printing. It might also "filter" some of the characters for special processing, looking for certain codes, such as the 12 code for top-of-form (more about that below). Each character is sent to a lower-level print-character subroutine which handles the actual communication with the printer, reads in the printer status (ready?, out of paper?, etc.) and sends the next character to be printed at the appropriate time.

Problems Department: When the printer driver encounters a top-of-form character (12), it attempts to space lines to the top of the next page. The 12 code that the driver *thinks* is top-of-form may actually be a numeric value within an escape sequence. In addition to the top-of-form problem, the printer driver also changes a line-feed code (10) to a carriage return code (13), causing problems for certain escape sequences. Another problem is that the printer driver might automatically cause a new line after a certain number of characters have been sent out for each line—this may be an undesirable action on some escape sequences. We'll describe how to circumvent these problems later.

Printer Hint

MORE ABOUT BUFFERS AND SPOOLING

Printer buffers are typically only a few thousand bytes long, each byte holding one character to be printed (or a control code). Radio Shack and other manufacturers sell *hardware print spoolers* which extend the few thousand bytes into tens of thousands. Radio Shack calls theirs a *Printer Controller* (PTC-64, part number 26-1269).

The hardware print spooler is connected between the computer and printer. It accepts characters to be printed as fast as the computer can pump them out, storing the characters in memory. At the same time, the spooler is handling the printer communication and sending characters from the other end of the storage—the first characters received are the first printed. If a 20-page report is to be printed, the spooler stores the 10,000 characters in memory and keeps on printing the report. Meanwhile, the computer has been fooled into thinking that the printer has received and printed the 10,000 characters, so it goes on to the next task. The spooler, therefore, frees up the computer to continue on another task, avoiding long waits while data is being printed.

The same approach is used on some Radio Shack operating systems to spool the data to be printed under *software* control. In this approach, a large portion of memory is set aside as a buffer to hold characters to be printed. The operating system sends characters to this buffer instead of the printer. A *background* task continually checks the buffer, sending characters to the printer if the buffer contains data. The background task runs simultaneously with normal processing, such as a BASIC program. Here again, the computer thinks it's done with the printer and goes on to other things, freeing up the system.

the printer in 50 separate actions in the interpreter, but to the printer it looks like a fairly constant stream of characters.

Note: The printer driver in your system is not usually compatible with some of the *escape sequences* we'll be discussing in this book. By not compatible, we mean that occasionally strange things will appear to happen (usually new line action) when you attempt to use some of the escape sequences. We'll define what the problems are and how to overcome them as we describe each printer function, so don't be too concerned about compatibility at this point.

Each time the printer receives a character it starts the printing process. The computer executes hundreds of thousands of instructions per second and sends characters at the rate of tens of thousands per second, in parallel operation, or up to thousands per second in serial mode. To the computer, the printing process is incredibly slow. The computer could have executed many instructions from the time the printer receives the character and starts moving the dot-matrix pins or daisy-wheel hammer to print the character, until the character is actually printed.

All printers, therefore, contain *buffers* to overcome the problem of speed difference between the computer and the printing action. The buffer in the printer is a small amount of memory, usually a few hundred to thousands of characters long, which stores characters as fast as the printer receives them from the computer. The computer can send all 50 of the 50 characters in the preceding message in a short time and then go and do other things while the printer is printing.

In the preceding message, the computer can send the 50 characters in about 1/2000 of a second in parallel mode. However, the fastest rate at which the characters can be printed ranges from about 1/3 of a second for a fast printer to three seconds for a slow printer. (In serial mode, the character transmission speed is more evenly matched to the printer speed. It takes about 5/12 of a second to send the 50 characters at 1200 bits per second.)

What if thousands of characters are being printed? Is it possible to send so many characters so fast that the printer buffer fills up? If the preceding BASIC line is changed to:

```
100 PRINT "THE QUICK PROGRAMMER . . . ENGINEER"
110 GOTO 100
```

the printer buffer would soon fill up. When full, the printer sends a signal back to the computer that the buffer is full, and the computer waits before sending more characters. (You wouldn't be aware of the signal, but the "printer-driver" program would be.) As soon as the printer emptied some of the buffer, the computer would send more characters.

You don't have to know how the buffer in the printer is used, but it is an important part of the printer and causes some otherwise unexplained actions—things such as a portion of a previous line being printed when you restart a program, or printing after the program has stopped.

CHAPTER 4

ALL THE CHARACTERS FIT TO PRINT

This chapter will describe the different character sets printed by Radio Shack printers. Most conform approximately to ASCII *characters, a standardized set of alphabetic, numeric, and special characters. Others include block graphics characters, Japanese and European characters, and special control code characters.*

WHAT ASCII IS

ASCII stands for "American Standard Code for Information Interchange" and was established to create a uniform way of representing the alphabetic characters from uppercase *A* through *Z*, lowercase *a* through *z*, the digits 0 through 9, and special characters such as @, $ and &. ASCII is an old code in terms of computing age. It was established when one of the primary printing devices was a teletypewriter. Teletypewriters (many are still in use) use paper tape as a storage medium instead of magnetic tape or disk. The paper tape uses eight columns along the tape to represent one frame of data, equivalent to one character, as shown in Figure 4-1.

The original ASCII code used *seven* bits represented by punched holes in the tape (one bit was always unpunched), and this seven-bit code carries over to the ASCII used today in Radio Shack printers. Table 4-1 shows the original ASCII code.

The first 32 characters, from decimal 0 through decimal 31, are *control characters*. These are non-printing characters that perform special functions. The line-feed and carriage-return characters discussed in a previous chapter fall into this group, as do such characters as form feed. Most of the characters in this portion of the ASCII code were primarily designed for telephone communications and really don't have specific functions in printers.

Printer Hint

MORE ON CONTROL CODES

The communications heritage of the control codes can be seen in their mnemonics. Code 1—an SOH—is a "Start of Header"; code 4—an EOT—is an "End of Transmission," and so forth. Other codes are more prosaic— VT is Vertical Tab, FF is Form Feed, and BS is Backspace. Code 127, while not in the Control Code group is an "all ones" in binary. This code is a Delete code and was used to obliterate all seven punch positions in a paper tape by punching all holes! So much of computer technology has roots from thirty years ago...

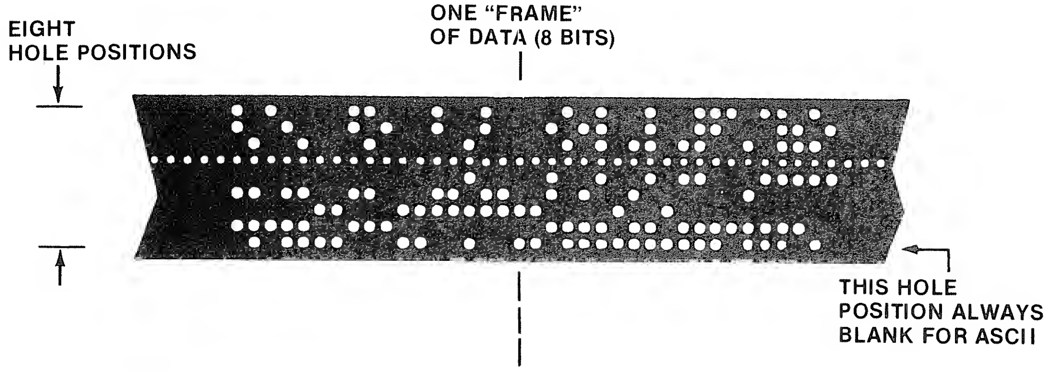


Figure 4-1. Paper Tape Representation

Table 4-1. Original ASCII Code

NUL Null (All Zeros)
SOH Start of Heading
STX Start of Text
ETX End of Text
EOT End of Transmission
ENQ Enquiry
ACK Acknowledgement
BEL Bell, or Attention Signal
BS Backspace
HT Horizontal Tabulation
LF Line Feed
VT Vertical Tabulation
FF Form Feed
CR Carriage Return
SO Shift Out
SI Shift In
DLE Data Link Escape

DC1 Device Control 1
DC2 Device Control 2
DC3 Device Control 3
DC4 Device Control 4
NAK Negative Acknowledgement
SYN Synchronous/Idle
ETB End of Transmitted Block
CAN Cancel (Error in Data)
EM End of Medium
SUB Start of Special Sequence
ESC Escape
FS File Separator
GS Group Separator
RS Record Separator
US Unit Separator
Sp Space
DEL Delete

First Hexadecimal Digit (columns); Second Hexadecimal Digit (rows)

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Control Codes (000–001); Special and Numeric (010–011); Alphabetic and Special (100–101); Alphabetic and Special (110–111)

Note: Most Radio Shack software printer drivers change a line-feed character (10) into a carriage return character (13). This results in occasional problems in certain escape sequences. We'll discuss the problems as we come to them.

The remaining 96 characters in the ASCII chart are all printable characters. Note that the first code (32) is a space, followed by a group of special characters, followed by the digits *0* through *9*, followed by another set of special characters, followed by the uppercase alphabetic characters *A* through *Z*, followed by another set of special characters, followed by the lowercase letters *a* through *z*, followed by another set of special characters.

There's a logical grouping to the codes, even though there are a lot of special character groups.

Over the years, manufacturers have usually followed the printable portion of the ASCII set, occasionally throwing in their own characters for some of the less frequently used characters. In the control code area, the basic carriage return, line feed, form feed (top-of-form), the *escape character*, and a few others have been used regularly by many manufacturers.

The escape character (code 27) has been standardized as a character to be used for starting special escape sequences, which define bold printing, italics, superscripting, subscripting, and the like. Unfortunately, because every printer differs in its capabilities, there is not a great deal of standardization among manufacturers in defining the escape sequences.

RADIO SHACK ASCII USE

Radio Shack, like other manufacturers, has modified the basic ASCII codes with its own set of characters. This set of characters is almost identical to the printable ASCII set—it's called the Tandy character set or modified ASCII. Whereas codes 91 through 94 print [, \ ,], and ∧ in ASCII, the equivalent Tandy characters are ↑ , ↓ , ← , and → . Newer Radio Shack printers can be switched between standard printable ASCII and the Tandy set, with the standard ASCII being the default. Older Radio Shack printers generally use the Tandy character set. The standard printer character set is shown in Table 4-2, along with the exceptions for various printers.

CODE SEQUENCES

The control code area, from decimal 0 through decimal 31, has been standardized in recent Radio Shack printers. All printers from about the LPVIII have used this standardized control code sequence. The purpose of the standardization is to make programs work the same on all printers when printing data, underlining, doing subscripting, bold printing, and so forth. The current definitions for control codes are shown in Table 4-3. Explaining the use of these control codes is what the majority of this book is about!

Note that many codes start with the escape character, but other codes use only a single character. These characters are sent to the printer the way any other characters are, but since they aren't printable or displayable, or can't be (in many cases) entered from the keyboard, special techniques must be used to include them in BASIC programs, and we'll be describing those in the next chapter.

THE REMAINING 128 BYTES

If you'll think about the ASCII codes for a moment, you'll notice that there are 128 codes, because there were seven columns of holes. (With one hole we could have two codes: 0 or 1; two holes: four codes 00, 01, 10, and 11; three holes: eight codes; four holes: sixteen codes; five holes: 32 codes; six holes: 64 codes; and seven holes: 128 codes.) However, 128 codes are only half of the total number of codes we could hold in the eight bits per byte of all Radio Shack computers. In eight bits, we could hold 256 codes. This means that instead of 128 characters from 0000000 (0) through 1111111 (127), we can get 256 characters from 00000000 (0) through 11111111 (255). If binary isn't your forte, don't worry about that calculation; but it is

Table 4-2. Tandy (Modified ASCII) Vs. ASCII Characters

Code Dec.	Code Hex.	Char.	Code Dec.	Code Hex.	Char.	Code Dec.	Code Hex.	Char.
32	20	(Space)	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	F	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	—	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	T	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	;	91	5B	[(↑)	123	7B	{
60	3C	<	92	5C	\(↓)	124	7C	!
61	3D	=	93	5D	](←)	125	7D	}
62	3E	>	94	5E	^(→)	126	7E	∿
63	3F	?	95	5F	—	127	7F	①

① ON CGP-115
ON QP-II

Parentheses characters are "Tandy" or "Modified ASCII"

Printers Using Tandy Set: LPII (modified), LPIII (modified), LPV (modified), QP-II

Printers Using ASCII Set: LPI (uppercase only), LPIV, LPVI, LPVII, QP-I, TP-10, CGP-115, CGP-220, DMP-100, DMP-110, DMP-120, DMP-200

Printers in Which One or Other Set May Be Selected: DMP-400, -420, -500, -2100, LPVIII

Daisy-wheel printers generally use ASCII wheels

true that eight bits can define 256 unique values from 0 through 255.

We could use the decimal values from 128 through 255, then, to define some additional characters. However, since we've covered all printable characters, what is left? It pretty much depends upon the printer and when it was designed

Early Radio Shack printers, like the LPI and LPII, don't use any additional characters. Starting with the LPV, however, the additional values are defined as European or special characters, such as a German umlaut or British pound sign. And since Radio Shack computer systems had some graphics capability, special block graphics were also defined. In addition, printers like the LPVII (later the DMP-100) used the upper 128 characters as graphics characters.

Printer Hint

BINARY NUMBERS

Binary numbers are not all that difficult to understand. On the other hand, they're not really necessary for most printer operations, so don't feel that you must learn them. We've put in an appendix to help you translate from decimal to binary and hexadecimal for all the values that really matter with printers—0 through 255 decimal (00 through FF hexadecimal, or 00000000 through 11111111 binary). Binary numbers are most often used in coding graphics data or in external programming mode in proportional spacing on daisy-wheel printers. If you're going to be doing a lot of those operations, it might behoove you to look into binary operations. One book is "Microcomputer Math" by William Barden, Jr., Howard W. Sams & Co., Publisher. Commercial!...

Table 4-3. Control Codes

8	Backspace one character on older daisy wheels
8,n	Backspace n pnts/microspaces on newer printers
10	Line feed, execute, most printers
12	Top of form, execute, most printers
13	Carriage return, execute, most printers
14	Underline end, most printers
15	Underline start, most printers
18	Select graphics, newer dot-matrix
19	Select data processing, new dot-matrix
20	Select word processing, new dot-matrix
26	Carriage return, execute, some printers
27,1 - 27,9	Proportional space 1–9 points or microspaces, newer printers
30	End graphics mode, newer dot-matrix
31	Elongation start, older printers
32	Elongation end, older printers
27,10	Set full reverse line feed, newer printers
27,14	Elongation/pitch start, newer printers
27,14	Select condensed type, older printers
27,15	Elongation/pitch end, newer printers
27,15	Cancel condensed type, older printers
27,17	Select proportional type, newer printers
27,18	Select 10 pitch, correspondence type, newer dot-matrix
27,19	Select 10 pitch, standard type, newer printers
27,19	Select graphics, older printers
27,20	Select condensed type, newer dot-matrix
27,20	End graphics mode, older printers
27,21	Set carriage return only, some daisy wheels
27,22	Reset carriage return only, some daisy wheels
27,23	Select 12 pitch standard type, newer dot-matrix
27,24	Daisy-wheel parameters, start, newer daisy wheels
27,25	Daisy-wheel parameters, end, newer daisy wheels
27,26	Set 1/8th forward line feed, some daisy wheels
27,28	Set 1/2 forward line feed, most printers
27,29	Select 12 pitch, correspondence type, several printers
27,30	Set 1/2 reverse line feed, newer printers
27,30	Bold end, older printers
27,31	Bold start, newer dot-matrix
27,32	Bold end, newer dot-matrix
27,32	Set 1/2 reverse line feed, older dot-matrix
27,49	Set 1/20 forward line feed, several printers
27,50	Set 1/12 forward line feed, newer dot-matrix
27,51	Set 1/36 forward line feed, several dot-matrix
27,51	Set twice forward line feed, old daisy wheel
27,52	Set 1 1/2 forward line feed, old daisy wheel
27,52,n	Set top of form to n, newer dot-matrix
27,54	Set full forward line feed, most dot-matrix
27,56	Set 3/4 forward line feed, most dot-matrix
27,66	Select italics, DMP-110
27,71	Set 4/5 forward line feed, several dot-matrix
27,77	Select microfont, DMP-110
27,73,n1,n2,d1,d2,d3	Print high-res graphics, several printers
27,138	Set full reverse line feed, old printers
28,n1,n2	Repeat print data, newer dot-matrix

Table 4-4. European and Special Characters

160	´
161	à
162	ç
163	£
164	`
165	µ
166	°
167	'
168	†
169	§
170	®
171	©
172	¼
173	¾
174	½
175	¶
176	¥
177	Ä
178	Ö
179	Ü
180	¢
181	~
182	ä
183	ö
184	ü
185	ß
186	™
187	é
188	ù
189	è
190	¨
191	ƒ

EUROPEAN AND JAPANESE CHARACTERS

Many Radio Shack printers are made in Japan. So it's reasonable to think that there may be Japanese character sets buried within the innards of the printer ROM or Read Only Memory. Also, many printers are marketed in Europe, so European characters are also likely candidates. The trend in later Radio Shack printers has been to make these character sets software selectable. Current Radio Shack printers come with either a Kana (Japanese phonetic) or European/special character set, or only the European/special set. The standard code values for the European/special characters for most

printers are shown in Table 4-4. (Some printers come with 64 characters; see "Printing European and Special Symbols" in Section II.)

Note that these character sets don't require any special escape sequence or other preparation—any time a character in the range of the values given in Table 4-4 is sent to a printer in which the set is included, the appropriate character is printed.

Printer Hint

MORE ON BLOCK GRAPHICS

Block graphics have been put into printers as a convenience for drawing simple graphic figures and ruled lines, either vertical or horizontal. They're a lot more convenient than normal graphics because they don't have to be coded as special values. You simply choose the proper block-graphics character from the table and print it, much as you would any other character in the printer-character set.

BLOCK GRAPHICS

Another set of standard characters found on many newer printers is *block-graphics* characters, shown in Table 4-5. Block-graphics characters are different from the usual graphics operations in printers because they are produced in normal text mode by sending a single character. Each produces any of 16 combinations of graphics blocks for a two-by-two matrix, line segments, or triangles. There is one qualification for using these characters, however. When they are used, the line feed for the printer must be set to less than the normal spacing. This can easily be done, and we'll show you how in another chapter. For now, take a look at Listing 4-1. It is a listing of a BASIC program that produces the form shown in Figure 4-2.

GRAPHICS PRINTING

The most common use of the upper 128 characters is in graphics mode to produce a graphics column of dots. In early Radio Shack printers, such as the LPVIII and DMP-100, a column of seven dots was printed by sending a character in the range of 128 through 255. The most significant bit in the character value was ignored, and the remaining seven bits defined the row in the column to be printed. This technique still applies. Characters

```
100 'DRAW FORM PROGRAM
105 CLEAR 1000
110 LPRINT CHR$(27);CHR$(28)
120 A=240: B=241: C=242: GOSUB 1000 'TOP LINE
130 A=245: B=224: C=245
140 FOR I=1 TO 4
150 FOR J=1 TO 5: GOSUB 1000: NEXT J 'SIDES ONLY
160 IF I<>4 THEN GOSUB 2000 'SIDES AND SHORT LINE
170 NEXT I
180 A=244: B=241: C=243: D=249: GOSUB 3000 'BEGIN BOX
190 A=245: B=224: C=245: D=245
200 FOR I=1 TO 6: GOSUB 3000: NEXT I 'VERTICAL LINES
210 A=244: B=241: C=250: D=249: GOSUB 3000 'LOWER BOX
220 A=245: B=224: C=245: D=245
230 FOR I=1 TO 63: GOSUB 3000: NEXT I 'VERTICAL LINES
240 A=246: B=241: C=248: D=247: GOSUB 3000 'BOTTOM LINE
250 LPRINT CHR$(27);CHR$(54)
260 END
1000 'SUB TOP OR SIDE LINES
1010 LPRINT CHR$(A);STRING$(59,B);CHR$(C)
1020 RETURN
2000 'SUB SIDES AND SHORT LINE
2010 LPRINT CHR$(245);STRING$(5,224);STRING$(25,241);STRING$(29,224);CHR$(245)
2020 RETURN
3000 'SUB ALL OTHER LINES
3010 LPRINT CHR$(A);STRING$(14,B);CHR$(C);STRING$(14,B);CHR$(C);STRING$(14,B);CH
R$(C);STRING$(14,B);CHR$(D)
3020 RETURN
```

Listing 4-1. BASIC Program for Form

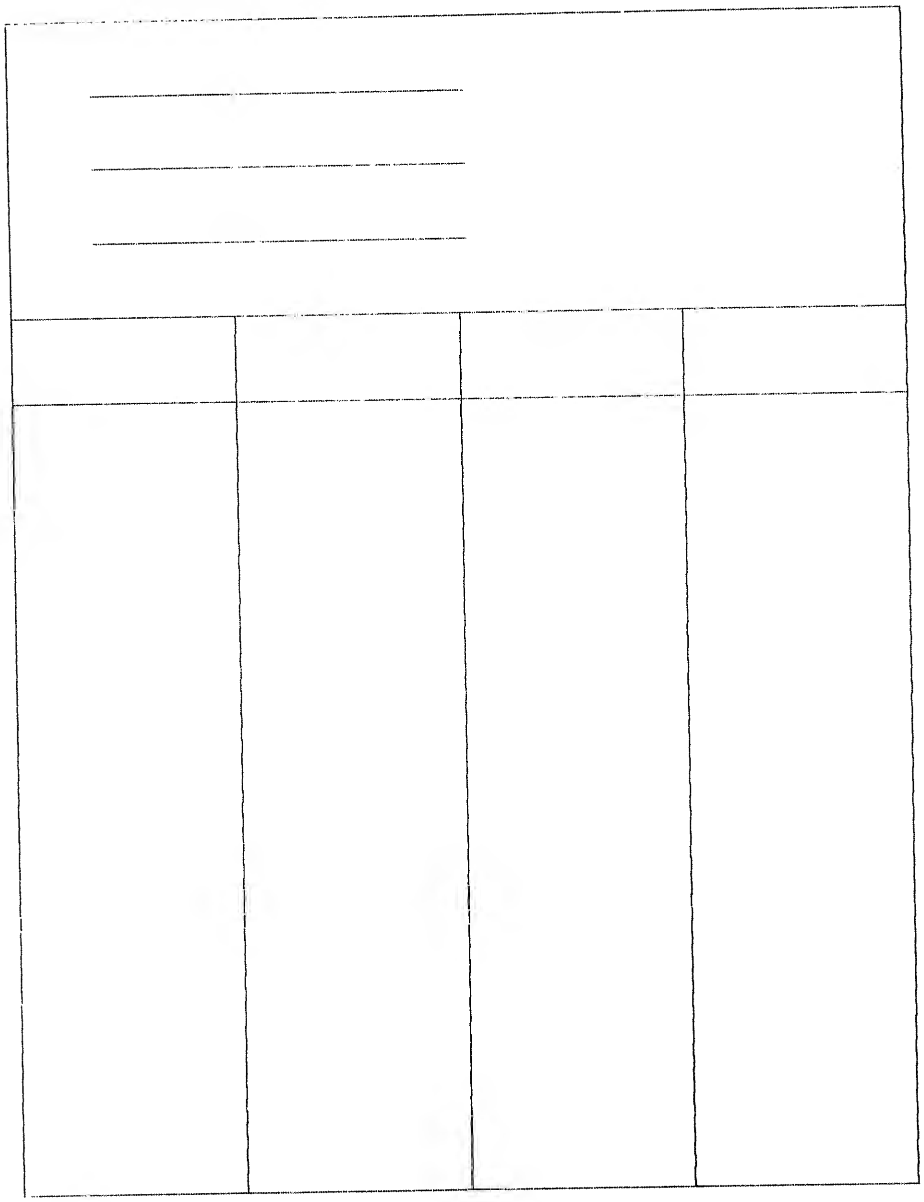


Figure 4-2. Form Done by Block Graphics

in the range of 128 through 255 are still graphics characters *when the graphics mode of the printer is set.* However, if the graphics mode isn't set, then the character in this range may be a European, Japanese, or block-graphic character.

As a sample of graphics printing, look at Figure 4-3. It shows a BASIC

Table 4-5. **Block Graphics Characters**

224	(blank)
225	
226	
227	
228	
229	
230	
231	
232	
233	
234	
235	
236	
237	
238	
239	
240	┌
241	─
242	┐
243	┬
244	├
245	│
246	└
247	┘
248	┴
249	┤
250	┼
251	
252	
253	
254	

program and resulting graphics picture produced on the DMP-2100. The graphics mode is set by LPRINT CHR$(18).

THE GENERAL CHARACTER SET OF NEW PRINTERS

In the newer Radio Shack printers, such as the DMP-120, DMP-420, and DMP-2100, then, there might be several different sets of characters. If the printer is in the data-processing or word-processing modes (really the same as far as the character sets are concerned) the printer will print normal text when sent a value in the 32 through 127 range of characters, will print European characters when sent a value in the 160 through 191 range of characters, and will print block graphics when sent a value in the 224 through 254 range of characters.

If the printer has been set to graphics mode, it will print a graphics row when receiving characters in the range of 128 through 255. Other characters will be ignored.

The control codes will not print but will cause some special action, if received in either the data-processing/word-processing modes, or in graphics mode.

There's another character to be considered that isn't defined in any of the before-mentioned character sets. When an undefined control sequence is sent, newer, smarter Radio Shack printers will print an hourglass shaped symbol like this X. This lets you know on the printout that an incorrect control code sequence or character was received. Another action the printer might take is to simply ignore the sequence or character. We'll discuss this in detail in a later section.

```
100 'COWBOY GRAPHICS PROGRAM
110 CLEAR 2000
120 LPRINT CHR$(18)
130 READ A
140 IF A=-1 THEN 190
150 IF A<>-2 THEN LPRINT CHR$(A);: GOTO 130
160 READ B,C
170 LPRINT STRING$(B,CHR$(C));
180 GOTO 130
190 LPRINT CHR$(30)
200 'COWBOY VALUES
210 DATA -2,50,128,-2,11,128,224,252,140,-2,4,128,-2,3,192,13
```

```
220 DATA -2,50,128,-2,10,128,142,159,191,248,240,242,246,246,255,255,223,134,134
,130,13
230 DATA -2,50,128,-2,13,128,248,-2,3,255,159,143,159,190,240,224,-2,4,128,192,2
24,192,224,224,192,224,192,192,13
240 DATA -2,50,128,-2,11,128,176,-2,6,255,254,254,255,255,252,253,254,-2,6,255,1
91,-2,3,159,-2,3,255,190,252,168,13
250 DATA -2,50,128,-2,7,128,224,240,248,252,254,-2,18,255,224,-2,3,192,128,128,1
29,159,143,159,13
260 DATA -2,50,128,240,248,252,188,142,130,159,191,-2,6,255,159,191,255,255,159,
131,-2,4,129,128,128,129,129,131,135
270 DATA 252,152,128,129,131,198,252,13
280 DATA -2,50,128,129,135,159,252,-2,5,128,243,255,143,131,129,128,135,143,-2,3
,136,152,240,224,-2,5,128,140,142,135,-2,3,128
290 DATA 131,131,129,13
300 DATA -2,50,128,-2,10,128,129,131,134,140,184,240,224,13
310 DATA -1
```

Figure 4-3. Sample Graphics Picture and Listing

CHAPTER 5

HOW TO TALK TO YOUR PRINTER

In this chapter we'll learn how to communicate with Radio Shack printers through BASIC and applications programs. We'll describe which BASIC commands are involved with all Radio Shack equipment. This won't be a detailed account of how to do specific functions on printers but will prepare you for the material in the second and third sections of the book.

BASIC PRINT COMMANDS

The fundamental BASIC command for communicating with a printer is the LPRINT command on the Model I, II, III, 4, 4P, 12, 16, Model 100, and Tandy 1000, 1200 and 2000. This is almost exactly equivalent to the fundamental print command on the MC-10 and Color Computer, the PRINT#-2, command. We'll give both versions in the following examples.

DIRECT MODE

The simplest way to print something on your printer is to enter an LPRINT or PRINT#-2 command in the *direct mode* in BASIC. If you wanted to print the message "THIS IS A TEST", you'd follow this procedure:

Turn on the computer and load BASIC. On Model I, III, 4, 4P, MC-10, and Color Computer systems without a disk, you're in BASIC as soon as you power up. You'll see a title message and then an "OK" or " > " prompt. On systems with a disk, such as the Model I, II, III, 4, 4P, 12, 16, Model 100 with disk, Color Computer with disk, Tandy 1000, 1200, and 2000, you may have to enter

BASIC

to load the BASIC interpreter from disk. Follow your operating system manual for this.

Now type the following line:

LPRINT "THIS IS A TEST" (I, II, III, 4, 4P, 12, 16, Model 100, Tandy 1000, 1200, 2000)
PRINT #-2,"THIS IS A TEST" (Color Computer, MC-10)

If your printer is turned on and is "on line" (indicated by an ON LINE indicator light on the front panel for most printers), you should see it print the message

THIS IS A TEST

After printing the message, BASIC will display an "OK" or " > " on the screen, indicating that it has stopped running and is waiting for the next command.

What you've done is execute a BASIC LPRINT (PRINT#-2,) command in the direct mode. In the direct mode, only a single line of BASIC is executed and it's executed immediately. This is different from a BASIC program where a number of different lines are executed.

Now try something else. Enter the following line:

```
LPRINT "THE NUMBER IS ";2.45 (I, II, III, etc.)
PRINT#-2,"THE NUMBER IS ";2.45 (CC, MC-10)
```

If the printer was ready for printing, you should have seen the message:

THE NUMBER IS 2.45

In this case the message enclosed in double quotes was printed, followed by the value of 2.45. Because there was both text and a constant value, you can see that more than one *item* can be printed with an LPRINT or PRINT#-2, command. The LPRINT or PRINT#-2, command can print as many items as you can fit onto a line.

PRINTING TEXT

Let's look at the first item—the text message. Printing text is the main task we'll be doing with our printer. Text is always enclosed in double quotes. The double quotes mark the text as a *string*, a fancy way of saying that there are printable text characters involved. Try the following:

```
LPRINT "THE FIRST IS ";2.45;" THE SECOND IS ";3.56
PRINT#-2,"THE FIRST IS ";2.45; "THE SECOND IS ";3.56
```

You should see this message printed on your printer:

THE FIRST IS 2.45 THE SECOND IS 3.56

Here we had four different items in the list, and they were printed one after the other. There were two text messages, or strings, enclosed in quotes, and two numeric values. The LPRINT or PRINT#-2, command always prints the items from left to right in the order they appear.

COMMAS VS. SEMICOLONS

Now try this command:

```
LPRINT "THE FIRST IS ",2.45," THE SECOND IS",3.56
PRINT#-2,"THE FIRST IS ",2.45," THE SECOND IS",3.56
```

Did you notice anything different in this printing? You should have noticed a wide space between the items—something like this:

```
THE FIRST IS          2.45          THE SECOND IS          3.56
```

The spacing will vary with the type of computer you have. The difference in the two sets of statements is that semicolons were used in one and commas were used in the other. When semicolons are used between print items, they tell the BASIC interpreter to avoid spaces between the items as they are printed. When commas are used between print items, they tell the

Printer Hint

MORE ON DIRECT MODE

You can execute any command to the printer in direct mode. Just code the proper string of characters and send them to the printer via an LPRINT (most systems) or PRINT#-2, (Color Computer and MC-10). If you want to print out a bold listing on a dot-matrix printer with bold print capability, you'd simply enter

```
LPRINT CHR$(27);CHR$(31)
```

and the bold mode would be set in the printer.

In direct mode, you can execute as many commands as you can cram into one string. You could even run short programs, as in

```
FOR I = 1 TO 1000: LPRINT I:
LPRINT I*I: NEXT I
```

which would print the numbers from 1 through 1000, and the squares of the numbers, on your printer.

BASIC interpreter to *tab*, or jump ahead, to predefined places along the print line. The tab positions are 0, 16, 32, and 48 for the Model I, III, 4 and 4P (Model III mode); 0, 14, 28, 42, and 56 for the Model II, 12, 16 and Tandy 2000; 0 and 16 for the Color Computer and MC-10; 0 and 14 for the Model 100; and 0, 8, 16, etc. for the Tandy 1000 and 1200.

You can intermix commas and semicolons any time you'd like. In this line:

```
LPRINT "A=";2,"B=";3,"C=";4
PRINT#-2,"A=";2,"B=";3,"C=";4
```

you'd get the print line

```
A= 2          B= 3          C= 4
```

with the number of spaces between the sets of values determined by your system.

CONSTANTS

Everything enclosed in double quotes is treated as text strings, but what about the numeric values in the LPRINT and PRINT#-2, lines? Anytime the BASIC interpreter finds an LPRINT item that is not surrounded by double quotes, it assumes it is not text to be printed. If the item is a numeric value, like 4, 5.66, 100000, or other values, it is printed more or less as it appears. We say more or less because there are exceptions to this. BASIC may convert some numeric values to slightly different forms before printing. The value 1.2222222222222222 will be printed in a shorter form, due to the accuracy of BASIC (1.22222222 in the Color Computer, for example) and the value 123334444444 will be printed in scientific notation (1.23334444E+11 in the Color Computer, for example). If you use constants in LPRINT and PRINT#-2, commands, expect them to be changed slightly if they are not whole integer numbers like 123, 67676, or 1000000, or if they contain a lot of digits. To print a constant exactly, surround it with double quotes to make a string out of it.

Notice in the examples above that BASIC also places a leading blank before positive values and a trailing blank after any value. This is just the way BASIC works, so be prepared to see this format on LPRINT or PRINT#-2, lines.

Printer Hint

ELIMINATING BLANKS AND OTHER GOODIES

If you'd like to get rid of leading blanks in printing numeric values, you must first convert the numeric value to a character string by an STR$ command. Having converted it to a string, you can then look at the first character of the string, by a LEFT$ command, and check to see if it is a blank. If it is, discard it. You can also check the number of characters in the string by the LEN command so that you can put figures in columns. Another good command is LPRINT USING, which will let you format strings to control the number of places in the printed strings. We can't give you a course in BASIC here (we've got our hands full with 30 printers...)—these are only clues to give you some idea of which commands can be used.

SIMPLE PROGRAMS

The simplest printer program is a single LPRINT or PRINT#-2, command. This program is just like the direct commands we were using in the above examples, except that the command now has a *line number* associated with it. Any time a line number is used, BASIC assumes that a program is being constructed and stores the line number and commands as a BASIC program line. Taking the case of the preceding example, let's construct a simple program:

```
100 LPRINT "THIS IS A TEST"
100 PRINT#-2,"THIS IS A TEST"
```

(Enter the first line for Model I, II, III, etc. and the second line for the Color Computer or MC-10.)

Unlike the BASIC commands executed in the direct mode, this command is not immediately executed. It's stored in memory as a BASIC program. To see it, enter a LIST after the BASIC "OK" or ">" prompt:

```
OK
LIST
100 LPRINT "THIS IS A TEST"
OK
```

To execute the program, enter a RUN command after the prompt. The RUN command means "Execute (run) the program from the first line of the program".

```
RUN
```

After the RUN, you should see the message THIS IS A TEST printed out on your printer, just as you did in the direct mode.

Let's try another one. Suppose you wanted to print out the text "A MAN, A PLAN, A CANAL, PANAMA!" on one line, followed by the text "READS THE SAME SDRAWKCAB!". This program would do the job:

```
100 LPRINT "A MAN, A PLAN, A CANAL, PANAMA!"
110 LPRINT "READS THE SAME SDRAWKCAB!"

100 PRINT#-2,"A MAN, A PLAN, A CANAL, PANAMA!"
110 PRINT#-2,"READS THE SAME SDRAWKCAB!"
```

You can see from the above that you can use as many LPRINT or PRINT#-2, commands in the program as you wish. Each one will print a single line of text on the printer. This program will print three lines:

```
100 LPRINT "THIS IS A SAMPLE OF TEXT"
110 LPRINT "PRINTED ON THREE LINES"
120 LPRINT "WITH A PRINTER"

100 PRINT#-2, "THIS IS A SAMPLE OF TEXT"
110 PRINT#-2, "PRINTED ON THREE LINES"
120 PRINT#-2, "WITH A PRINTER"
```

If the items in the print list are ended with a semicolon or a colon, a new line will not be started. This program will print the text above on a single line:

```
100 LPRINT "THIS IS A SAMPLE OF TEXT";
110 LPRINT "PRINTED ON THREE LINES";
120 LPRINT "WITH A PRINTER"

100 PRINT#-2, "THIS IS A SAMPLE OF TEXT";
110 PRINT#-2, "PRINTED ON THREE LINES";
120 PRINT#-2, "WITH A PRINTER"
```

The print line would look like this:

```
THIS IS A SAMPLE OF TEXTPRINTED ON THREE LINESWITH A PRINTER
```

Notice that the lines ran together because no spaces were left in between the three text strings. Note also that the last LPRINT or PRINT#-2, command had no semicolon after the list. If a semicolon had been placed after the list, the line would not have been printed on many printers *because most printers print the line only after a carriage-return character has been received.* BASIC automatically sends a carriage return after each LPRINT or PRINT#-2, that ends without a semicolon or comma.

If a comma is used to end a print list, BASIC will *tab the print position to the next tab without printing.* Further printing will proceed from the new tab posi-

Printer Hint

DELETING, INSERTING BASIC LINES

To delete any BASIC line, just type in the line number with no other characters. To delete line 100, for example, type

```
100
```

followed by <ENTER>.

To insert a BASIC line, use a line number that's intermediate between the two line numbers around the insertion point. To insert the line LPRINT "YOU, TOO, CAN WRITE REAL GOOD" between

```
110 LPRINT
    "W. BARDEN, JR"
120 LPRINT "REPORT 1"
```

you could use

```
115 LPRINT "YOU, TOO, CAN
    WRITE REAL GOOD"
```

The result would be

```
110 LPRINT
    "W. BARDEN, JR."
115 LPRINT "YOU, TOO, CAN
    WRITE REAL GOOD"
120 LPRINT "REPORT 1"
```

tion. If a following LPRINT or PRINT#-2, is not done after a terminating semicolon or colon in a print list, the entire line or a portion of a line may be lost.

An LPRINT or PRINT#-2, alone simply prints a line without anything on it. In fact, this causes BASIC to send only a carriage-return character alone. To see how this works, look at the program below. It is a complete program to print mailing labels in five lines!

```
100 LPRINT "William Barden, Jr."
110 LPRINT "200 N.S. Memory Drive"
120 LPRINT "Computer City, CA. 92692"
130 LPRINT: LPRINT: LPRINT
140 GOTO 100
```

```
100 PRINT#-2, "William Barden, Jr."
110 PRINT#-2, "200 N.S. Memory Drive"
120 PRINT#-2, "Computer City, CA. 92692"
130 PRINT#-2: PRINT#-2: PRINT#-2
140 GOTO 100
```

There are several things that may not be obvious in the preceding example: More than one BASIC command can be put on a line. When this is the case, a colon is used to separate the commands. The GOTO command in BASIC is used to jump to a line, and in this case, causes the same sequence of BASIC lines to be executed over and over. Press the <BREAK> key to stop the program before you're up to your eyeballs in mailing labels!

This program prints three separate lines for the mailing labels, prints three blank lines (by the LPRINT or PRINT#-2, statements), and then repeats the whole procedure again and again. The spacing is set for six lines-per-inch and spacing between labels of one inch.

VARIABLES

The LPRINT or PRINT#-2, item list can have text in the form of strings and constants as we saw above. It can also have *variables*. Variables are BASIC quantities that are assigned names of one or two characters (or more in Model IV and in MS-DOS BASICs). The value for the quantity may be changed within the program. It's like keeping the combination to a wall safe inside a shoebox. You've printed "WS" on the outside of the box to stand for Wall Safe. Inside the box is a slip of paper with the combination of the wall safe. You can redo the combination at any time and store a new value inside the WS box on the slip of paper. The WS box is the variable and the slip of paper is the current value of the variable.

Let's see how variables are printed. The program

```
100 A = 34.56: B = -56.89
110 LPRINT "VALUE OF A = ";A;"VALUE OF B = ";B
100 A = 34.56: B = -56.89
110 PRINT#-2,"VALUE OF A = ";A;"VALUE OF B = ";B
```

will print

```
VALUE OF A =  34.56 VALUE OF B = -56.89
```

The value of the A variable was printed after the first text message, followed by the second text message, followed by the value of the B variable. Semicolons were used so that no spaces were used other than

Printer Hint

MORE ON VARIABLES

Variable names are two characters in earlier systems (up to the Model III) and more than two characters in later systems, such as the Model IV and Tandy 2000. We'll use two characters in all of the programs in this book to make the code compatible with all systems. The first character in variable names must be an alphabetic character. Other characters can be alphabetic or numeric. You can use any combination of characters for a variable name, except for certain reserved system names.

There are two main types of variables—numeric and string variables. We'll be dealing with both in the programs and code in this book. Numeric variables can hold numeric values from very small numbers to very large numbers. Just set the variable equal to a numeric quantity by an equals sign (AA = 1.234; BV = 100000; C = 2). Depending upon the computer system, there are also subclasses of numeric variables. In spite of this complexity, however, you should have no problem if you just stay with simple numbers, the way you normally write them (but without commas).

those put in for the variables. You can see that a variable is printed the same as a constant, with a leading space or minus sign and a trailing space.

Another type of variable is the *string variable*. This variable can hold a constant text string, or can be used to hold different text strings. The following program uses the string variables "A$", "B$", "C$" to print the same three-line message as before:

```
100 A$ = "THIS IS A SAMPLE OF TEXT"
110 B$ = "PRINTED ON THREE LINES"
120 C$ = "WITH A PRINTER"
130 LPRINT A$: LPRINT B$: LPRINT C$
140 LPRINT A$;B$;C$
```

```
100 A$ = "THIS IS A SAMPLE OF TEXT"
110 B$ = "PRINTED ON THREE LINES"
120 C$ = "WITH A PRINTER"
130 PRINT#-2,A$: PRINT#-2,B$: PRINT #-2,C$
140 PRINT#-2,A$;B;C$
```

Line number 130 prints out the three lines as before—it's really three separate LPRINT or PRINT#-2, commands. Line 140, however, prints out something completely different. Because there's only one LPRINT or PRINT#-2, command, line 140 prints out the three string items, which come out as:

```
THIS IS A SAMPLE OF TEXTPRINTED ON THREE
LINESWITH A PRINTER
```

MORE ON STRING VARIABLES

String variables are used all the time in LPRINT and PRINT#-2, commands. There are a number of string-related functions in BASIC that are used to make string handling easier and that also help in defining LPRINT or PRINT#-2, strings. One of these is the CHR$() function.

CHR$() is used to create a one-character string from any value. We saw earlier how string constants could define a string to be printed, so why is CHR$() needed? Even though a one-character string could be defined and printed, as in:

```
100 LPRINT "&"
100 PRINT#-2,"&"
```

which would print the single-character string "&", this method isn't adequate for all single-character strings that are required. What about printing a control code from the 32 characters that are not displayable? How would you send a carriage return, control code 13, to the printer, for example?

Most control codes can't be displayed or produced from the keyboard, and that's where CHR$() comes in. In addition, graphics characters, European characters, and block graphics also cannot always be produced from the keyboard, and CHR$() is used for those characters as well. CHR$() will produce a one-character string equal to a value inside the parentheses. The one-character string can be combined with a text string to produce a combination string.

To see how this works, let's use CHR$() to produce a line of text, five blank lines, and another line of text:

```
100 A$ = "THIS IS THE FIRST LINE OF TEXT"
```

```
110 B$="THIS IS THE SECOND LINE OF TEXT"
120 LPRINT A$ +CHR$(13)+CHR$(13)+CHR$(13)+CHR$(13)
     +CHR$(13)+B$
```

```
100 A$="THIS IS THE FIRST LINE OF TEXT"
110 B$="THIS IS THE SECOND LINE OF TEXT"
120 PRINT#-2,A$ +CHR$(13)+CHR$(13)+CHR$(13)+CHR$(13)
     +CHR$(13)+B$
```

In line 120, we used a carriage return control code character, a decimal 13, as shown in Table 4-3. The actual string created in line 120 was made up of 66 characters, 30 from the A$ string, 31 from the B$ string, and the five decimal 13 values, as shown in Figure 5-1.

|←30 CHARACTERS→|
A$ = "THIS IS THE FIRST LINE OF TEXT"

|←31 CHARACTERS→|
B$ = "THIS IS THE SECOND LINE OF TEXT"

A$ + CHR$(13) + CHR$(13) + CHR$(13) + CHR$(13) + CHR$(13) + B$

30 CHARACTERS + 5 CHARACTERS + 31 CHARACTERS

Figure 5-1. Combining Strings for Printing

All of the characters were sent to the printer when the LPRINT or PRINT#-2, command was executed. In addition, a carriage return was added at the end of the print, as BASIC does when an LPRINT or PRINT#-2, item list doesn't end with a semicolon or comma. The result on the printer was:

```
THIS IS THE FIRST LINE OF TEXT





THIS IS THE SECOND LINE TEXT
```

The five carriage returns between the text caused five new lines, as we've explained in previous chapters.

The CHR$() function can be used to produce *any* values to be sent to the printer. Any eight-bit value from 0 through 255 can be produced—even displayable characters. The program

```
100 LPRINT CHR$(65)+CHR$(66)+CHR$(67)
100 PRINT#-2,CHR$(65)+CHR$(66)+CHR$(67)
```

prints the text ABC on the line printer.

The most important use of CHR$(), however, is in producing control codes and *escape sequences* to be sent to the printer for special functions. An example is the following program, which uses CHR$() to set underlining on many printers.

```
100 LPRINT "THIS IS NOT UNDERLINED"
```

```
110 LPRINT CHR$(15);
120 LPRINT "BUT THIS IS"

100 PRINT#-2,"THIS IS NOT UNDERLINED"
110 PRINT#-2,CHR$(15);
120 PRINT#-2,"BUT THIS IS"
```

Note that a semicolon followed the CHR$(XX) to prevent a new line. The string could have been:

"THIS IS NOT UNDERLINED" + CHR$(15) + "BUT THIS IS"

We'll be seeing a lot of the CHR$() function in sections two and three of this book, where we'll be using special control codes and escape sequences for many different functions.

COLOR COMPUTER AND MC-10 USERS ONLY—LOWERCASE

Up to this point, most of our examples have used uppercase, or capital letters. To produce lowercase letters on the Color Computer or MC-10, just hold down <SHIFT>, followed by *0*. Characters entered after that will be in inverse video on the screen. This will indicate that they are lowercase (small letters). Figure 5-2 shows how the text "TWEEDLEDUM and tweedledee" appears on the Color Computer or MC-10 screen. To get back to uppercase, press <SHIFT> followed by *0* again. We'll use both upper- and lowercase in the examples from now on, so don't think that your printer will only be able to print in uppercase.

UPPERCASE — T W E E D L E D U M A N D

LOWERCASE — T W E E D L E D U M

Figure 5-2. Color Computer/MC-10 Upper and Lower Case

BASIC PRINTER DRIVER CAUTIONS

BASIC programs are generally a lot easier to work with than other programs when it comes to using printer functions. However, as we've mentioned before, BASIC uses a software line-printer driver that may create problems in certain escape sequences. We'll let you know what the problems are as we discuss each line printer function. However, let's start with a definition of what kinds of problems to expect. They're not catastrophic, but they may cause baffling results, unless you know what's happening.

- The BASIC line-printer driver intercepts all line-feed (10) characters and changes them to carriage returns (13). The rationale for this is the old "Radio Shack line feed/carriage return with just a carriage-return

character" which we discussed earlier. When a 10 character is found *anywhere* in characters to be printed, it will be changed to a 13 character. Some escape sequences include numeric values, and if one of those numeric values is a 10, it will be changed to a 13!

- A top-of-form character (12) causes a page-eject action based on an internal line count in the software driver. If a 12 character is used inside an escape Sequence, it will be detected as a top-of-form character by the printer driver. This will cause a series of line feeds to space printer paper to the top of what the printer driver thinks is the next page.
- The printer driver contains a variable that sets the number of characters per line. Normally, this would cause a new-line action after the line-printer driver thought the right edge of the paper was close. However, if a large number of backspacing characters are sent, or if a large number of graphics characters (which have hundreds of positions per line) are sent, making the character count for the current line appear to exceed the maximum number per line set in the printer driver, the printer driver will send a carriage return before the end of the line is printed.

What alternatives do you have in working with the BASIC printer driver? You could write your own driver, but then it becomes a chore to figure out where to put the driver in all types of systems, for all operating systems, and for all configurations of memory. The alternative chosen for this book is to work with the BASIC driver and to point out areas in which problems may occur. Usually the problems are surmountable.

CHAPTER 6

WHAT KIND OF PRINTING CAN YOU DO WITH YOUR PRINTER?

In this chapter we'll expand upon the material presented in Chapter 4—the control codes for different functions on your printer. The control codes determine what basic functions can be performed, but they don't define many of the other functions that can be done by combining sequences of control codes. For example, many printers have a control-code sequence for printing a single column of graphics, but you can use a series of these to print out a complete graphics picture. In this chapter we'll give you the actual control codes for your specific printer; in Section II we'll show how to use the sequences for text printing applications. In the final section we'll show you how to use the sequences for graphics applications.

THE MASTER TABLE

Table 6-1 at the end of this chapter is a master list of control-code and other code sequences, cross-referenced by printer type. The printer types are listed across the top of the chart and are grouped as follows: all Roman numeral line printers, all DMP (dot-matrix) printers, all daisy-wheel printers, the CGP-115 printer/plotter, the CGP-220 ink-jet printer, all thermal printers, and the early Printer/Plotter.

The left-hand side of the chart lists all functions by general type—all line feed–related functions are grouped together, for example. In some cases there will be (old) and (new) sequences. "Old" refers to the older printers and "new" refers to the most recent printers. In some cases, newer printers have switch settings to emulate older printers—the DMP-500 has a switch to emulate the old Line Printer V codes, for example. This is denoted by a "J" at the junction of the appropriate function row and printer column.

If there's a dot at a particular function/printer junction, it means that the

printer can perform that function by the control code given. If there's no dot, it means that the printer can't use that control code.

The code for the function is given as a decimal number in the column at the left of the table. We haven't provided the hexadecimal equivalent because decimal will be most commonly used in a CHR$() command and works on all Radio Shack printers. If you'd like to use hexadecimal, consult the appendix for the decimal to hexadecimal conversions. You'll be able to use the prefix &H to specify hexadecimal in some Radio Shack computer systems, but not all.

The columns labeled DP, WP, and GR refer to the three modes found in newer Radio Shack printers: data processing, word processing, and graphics, as discussed in Chapter 2. Depending upon the printer, some of these modes will not be present. In newer "DMP" printers, you'll probably find that all modes are present—you'll be able to do fast, practical printing in data-processing mode, better looking printing in word-processing mode, and dot-graphics in graphics mode. In older "LP" printers there'll probably be only the data-processing option and no word-processing option, with possibly a graphics mode, as well. In some older printers, there may be only a text printing mode, which is essentially a data- and not word-processing mode. Daisy-wheel printers have only word-processing mode, as the printers are oriented towards quality printing with subscripting.

The dot, "P", or "M" under the DP, WP, and GR columns refers to the type of action that is taken when the command is sent to the printer. A dot means that the command is received and recorded, but that no printing or motion results. A "P" means that printing results after the command has been received, as in the case of the repeat command. An "M" means that some type of motion occurs without printing, such as positioning the print head to a certain print column.

You'll notice that most of the basic functions start with an escape character, a decimal 27, or a hexadecimal 1B. This is especially true in the newer printers. Sequences that do not start with an escape are primarily underlining, line feeds, and carriage returns.

FUNCTIONS

In the following discussion we'll describe the major categories of commands and how they work, in general terms. We'll go into excrutiating detail in the next two sections.

SELECT MODE

These commands generally are found on the newer generation of printers and select one of the three modes of printing—data processing, word processing, or graphics. A decimal 30 usually resets the graphics mode. In some of the printers, the command sequence is switch-selectable to make the newer printer compatible with an older model. Remember that the biggest difference between the data-processing and word-processing mode is in the line feed. In data-processing mode a line feed of less than a full line is generally not acted upon until the end of the line. In word-processing mode, a line feed of less than a full line is acted upon when it's received so that superscripting and subscripting can be done.

BOLD PRINTING

The next catagory of commands sets bold printing and is found in the newer

DMP printers. Bold printing can also be done with the daisy-wheel printers,. but by different means—we'll show you how in the next section.

ELONGATION

This function can be done on most printers. Older printers use the old elongation start and end commands, a single command instead of an escape sequence. The elongation command keeps the same character height but spreads out the dot matrix over twice the distance, horizontally. This results in a larger character—about twice normal size. The actual size of the character depends upon what pitch has been selected—an elongated character in "condensed" mode will be a different size than one in "standard" mode, for example. The elongate mode used in daisy-wheel printers doesn't elongate characters—it can't, because the characters are not dot matrix. What it does do, however, is to set a 12-characters-per-inch pitch. The standard pitch for daisy-wheel printers is 10 characters per inch, and this is reinstated after the printer gets a 27,15 code sequence. For daisy-wheel printers, therefore, the effect is to produce a more condensed printing, the opposite of the elongated printing action.

UNDERLINING

Underlining is set not by an escape sequence, but by the old control code for underlining, decimal 15. Underlining is found on most printers, and when it isn't there by a control-code function, it is possible to achieve it sometimes by repositioning and printing a graphics line.

LINE SPACING, LINE FEEDS, AND CARRIAGE RETURNS

Line spacing is set in many printers by sending an escape sequence. Depending upon the printer, different line spacings forward *and back* can be set. There's almost always a *half-reverse line feed* and a *half-forward line feed* because of superscripting or subscripting requirements. Also, in some cases, there's a special line-space setting for printing graphics which don't use the "empty space" between character lines. Line-space settings are given in Table 6-1, in line-feed units which are usually six lines per inch, that spacing being the historical standard and one that's commonly used for normal printing. A half-line feed is, therefore, 1/12 of an inch; a 3/4-line feed is 1/8 of an inch, and a 1/48-line feed is 1/288 of an inch.

In two older printers, line feeds of three lines per inch and four lines per inch are used—this comes out to a "times 2" and "times 1 1/2" of the standard 1/6 of an inch line feed, respectively.

It's important to note when a line-feed spacing is *set* and when it is *executed*. In data-processing mode, the line spacing is usually set by the line-spacing command, and executed when carriage-return or line-feed characters are sent to the printer. In word-processing mode, most line-spacing commands set the spacing, and the spacing is done at the time of the carriage return or line feed. However, half-line feed commands are usually executed as they're received, for subscripting, along with certain other spacing commands. In graphics mode, certain line spacing commands are also executed when they're received.

The carriage-return and line-feed actions can be very confusing. On newer printers, they work this way: The carriage-return action is usually set by a DIP switch so that a carriage return can result in either a carriage return and line feed, or only a carriage return. The normal setting for Radio Shack software is for both a carriage return and line feed. On daisy-wheel printers, a special escape sequence (27,21/27,22) can be sent, which is

Printer Hint

MORE ON BOLD

Bold printing is done automatically in dot-matrix printers. Once the command is sent to the printer, its electronics will print a character twice, to highlight the character in bold. Once bold mode is set, bold printing remains in force until the printer is reset.

In daisy-wheel printers, bold print is achieved by printing the characters, then executing a series of commands to backspace the print head, and reprinting the characters again over the first set. See "Using Bold Printing" in Section II.

similar to setting the DIP switch, and is used for overprinting or bold face. On older printers, the carriage-return action is also sometimes switch selectable, but this varies with the printer. The typical case for most older printers is a carriage return with line feed to get both a new line and a start at the beginning of the line.

The carriage return spaces the line in accordance with the most recent line-spacing command. If you've set 3/4-line spacing, for example, a carriage return will cause a 3/4-line space and not a full line space. Set the proper line spacing before a carriage return is issued.

The line-feed action is also determined by the printer model. On some newer models, a line feed can be switch selectable to cause a carriage return and line feed or a line feed only. The typical case is carriage return and line feed. In many cases Radio Shack software intercepts the line feed character and even though the printer is capable of performing a line feed without carriage return, the software does both a carriage return and line feed.

FORM FEED

The top-of-form escape sequence is used on newer printers only and is used to set the page length. After the page length is set, sending a form-feed character (decimal 12) will eject the proper amount of paper from the printer. However, in most cases Radio Shack software will intercept the form-feed character and issue a series of carriage returns instead, making the top-of-form action useless. Assembly language programs can circumvent this action.

TYPE FACE, PITCH

There are usually several typefaces and pitches that can be set on newer printers. In the daisy-wheel printers, of course, the typeface is determined by the actual wheel, and the pitch (number of characters per inch along the line) is set by the 27,14/27,15 codes. A pitch of 10 is called "pica," in typewriter terms. A pitch of 12 is called "elite," in typewriter terms. A pitch of 12 is more dense than a pitch of 10—you can get 96 characters in an 8-inch line versus 80 characters in the 10-pitch line.

Condensed characters are even closer together than 12-pitch characters. Typically they are 16.7 characters per inch, but this depends upon the printer.

In addition to allowing you to select the pitch, some printers also give you a selection of *fonts*. A font is a fancy word for appearance of the type, or the style. A plain font would be good for practical high-speed printing—this is called a standard font. A fancier font would be better for documents intended to impress—the font here is called "correspondence." An italicized type is offered on one printer, and the "microfont" of the DMP-110 is especially for subscripting.

Printer Hint

MORE ON FONTS

A font is a complete set of type in one style. There's a wide range of fonts to choose from when you have typesetting done commercially—literally hundreds and hundreds of different styles, from "script" to "computer" styles. As printers become powerful in terms of higher density printing, we'll see more and more type fonts implemented on printers, making the printer design a combination of electronics and typography and graphics arts.

HORIZONTAL POSITION

In most of the newer printers, and in some of the older graphics printers, you can position the print head to a *print position* along the line. This print position is based upon the number of dots making up the line, which is related to the number of dots making up a character. If a character is six dots wide, for example, an 80-character line would have 480 dot positions. The horizontal position escape sequence gives you the ability to position the print head anywhere along the line. You can do the same type of positioning in many daisy-wheel printers; but as there are no dot positions, the spacing is specified in "microspaces," from one to six or one to nine per character position.

DAISY-WHEEL PARAMETERS

Daisy-wheel printers have a special escape code sequence which is used with proportional printing daisy wheels. We'll show you how to do it in Section II.

SHEET FEEDING

More expensive printers have *sheet feeders,* mechanisms that feed the printer single sheets of paper, one at a time. The sheet feed is activated by an escape sequence.

REPEAT SEQUENCE

Many newer printers have a *repeat* sequence which allows you to repeat the same character many times. This is especially handy for graphics.

SPECIAL ACTIONS

The CGP-115 is the four-color plotter device that isn't really a standard printer at all. We've included it in this book because it's capable of printing text, and of performing a few other limited text operations. The CGP-115 commands relate to selecting text mode (from graphics), moving the paper in reverse, and rotating the pen holder to select one of four colors.

The CGP-220 is the color ink-jet printer. Although the actions are very similar to a normal dot-matrix printer, there are some special commands, relating to color selection and the proportions of characters, that aren't found in other printers.

The last group of functions can be classified in the "strange and unusual" bin. (We'll say "strange and unusual" only if you don't own a Quick Printer I, II, or Daisy-wheel I; in that case we'll say ...ah... "powerful.") These are very non-standard codes listed for completeness only.

Printer Hint

BEL CODE

The audio alarm is an interesting code. On old teletypewriters, there was a *BEL* code which rang a bell on the machine. A three-bell story, coming in over the wire service teletypewriter, was a "hot story." Since most microcomputers have a built-in speaker, the bell in the printer is not often used anymore, except for the QP-I.

OTHER PARAMETERS

For easy reference, we've also included other particulars in Table 6-1 relating to block graphics characters, European characters, Japanese Kana characters, and serial vs. parallel connections.

Table 6-1. Printer Master Table

					LPI	LPII	LPIII	LPIV	LPV	LPVI	LPVII	LPVIII	DMP-100	-110	-120	-200	-400	-420	-500	-2100	DW-I	DW-II,IIB	DWP-210	DWP-410	CGP-115	CGP-220	QP-I	QP-II	TP-10	PLOT/PRNT	
CODE	FUNCTION	DP	WP	GR	1	2	3	4	5	6	7	8	1	1	1	2	4	4	5	2	D	D	2	4	1	2	1	2	T	P	
18	Select graphics (new)	o	o		.	.	.	.	.	.	o	o	o	o	o	o	J	J	J	o	.	.	.	.	.	o	.	.	.	.	
27,19	graphics (old)				.	.	.	.	.	.	.	.	.	.	.	.	J	J	.	.	.	.	.	.	.	.	.	.	.	.	
19	data processing		o		.	.	.	.	.	.	.	o	.	o	.	o	o	o	o	o	.	.	.	.	.	.	.	.	.	.	SELECT MODE
20	word processing	o			.	.	.	.	.	.	.	o	.	o	.	o	o	o	o	o	.	.	.	.	.	.	.	.	.	.	
30	Graphics mode, end (new)			o	.	.	.	.	.	.	o	o	o	o	o	o	J	J	J	o	.	.	.	.	.	o	.	.	.	.	
27,20	, end (old)				.	.	.	.	.	.	.	.	.	.	.	.	J	J	.	.	.	.	.	.	.	.	.	.	.	.	
27,31	Bold start	o	o		.	.	.	.	o	.	.	.	.	o	.	o	o	o	o	o	.	.	.	.	.	.	.	.	.	.	
27,32	Bold end (new)	o	o		.	.	.	.	J	.	.	.	.	o	.	o	J	J	J	o	.	.	.	.	.	.	.	.	.	.	BOLD
27,30	Bold end (old)	o			.	.	.	.	J	.	.	.	.	.	.	.	J	J	J	.	.	.	.	.	.	.	.	.	.	.	
27,14	Elongation/pitch start (new)	o	o	o	.	o	.	o	.	.	.	o	.	o	o	o	J	J	J	o	Q	Q	Q	Q	.	o	.	.	o	.	
27,15	end (new)	o	o	o	.	o	.	o	.	.	.	o	.	o	o	o	J	J	J	o	N	N	N	N	.	o	.	.	o	.	ELONGATION
31	Elongation start (old)	o			.	.	o	.	o	o	o	.	o	.	.	.	J	J	J	.	.	.	.	.	.	.	.	.	.	.	
32	end (old)	o			.	.	o	.	o	o	.	.	.	.	.	.	J	J	J	.	.	.	.	.	.	.	.	.	.	.	
15	Underline start	o	o		.	.	.	o	o	.	.	o	o	o	o	o	o	o	o	o	.	o	o	o	.	.	o	.	.	.	UNDERLINE
14	end	o	o		.	.	.	o	o	.	.	o	o	o	o	o	o	o	o	o	.	o	o	o	.	.	o	.	.	.	
27,54	Line feed,forward,set, full	o			.	.	o	.	o	o	.	o	.	o	o	o	o	o	o	o	o	.	.	.	.	o	.	.	.	.	
27,71	, 4/5	M	M	M	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	
27,56	, 3/4	o	M		.	.	o	.	o	o	.	o	.	.	o	o	o	o	o	o	.	.	.	.	.	o	.	.	.	.	
27,28	, 1/2	o	M		.	.	.	o	o	o	.	o	.	o	o	o	o	o	o	o	.	o	o	o	.	.	.	.	.	.	
27,26	, 1/8		o		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	.	
27,50	, 1/12		o		.	.	.	.	.	.	.	.	.	.	o	o	o	o	o	.	.	.	.	.	.	.	.	.	.	.	
27,49	, 1/20	M	M	M	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	LINE SPACING, LINE FEEDS, CARRIAGE RTNS
27,51	, 1/36		o		.	.	.	.	.	.	.	.	.	.	.	.	o	o	o	.	.	.	.	.	.	.	.	.	.	.	
27,52	, 1 1/2		o		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	
27,51	, twice		o		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	
27,10	Line fd,rvrse,set,full(new)	o	M		.	.	.	.	.	.	.	o	.	.	.	o	o	o	o	o	.	o	o	o	.	.	.	.	.	.	
27,138	,rvrse,full (old)	M			.	.	.	o	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
27,30	,rvrse,set,1/2(new)	o	M		.	.	.	o	.	.	.	o	.	o	.	o	J	J	J	o	.	o	o	o	.	.	.	.	.	.	
27,32	,rvrse,set,1/2(old)	o			.	.	.	.	.	.	.	.	.	.	.	.	J	J	J	.	.	.	.	.	.	.	.	.	.	.	
27,21	Carriage return only, set		o		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	o	o	.	.	.	.	.	.	
27,22	, reset		o		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	o	o	.	.	.	.	.	.	
10	Line feed, execute	M	M	M	o	o	D	D	B	B	A	GB	A	I	B	DH	BG	BG	BG	G	A	A	I	I	I	I	o	.	I	I	
13	Carriage return, exec (new)	M	M	M	o	o	o	E	C	C	A	HC	A	A	CH	CH	CH	BH	CH	H	A	R	R	R	.	A	o	o	A	A	
26	Carriage return, exec (old)	M		M	.	.	.	.	.	.	E	.	E	E	.	.	.	.	.	.	.	.	.	.	.	.	.	.	E	.	
27,52,n	Top of form, set to n	o	o	o	.	.	.	.	.	.	.	.	.	o	.	o	o	o	o	o	.	.	.	.	.	.	.	.	.	.	FORM FEED
12	, execute	o	o	o	F	.	.	.	.	.	.	.	.	o	.	o	o	o	o	o	.	.	.	.	.	.	F	.	.	.	
27,17	Type, proportional, select	o	o		.	.	.	o	.	.	.	o	.	o	.	o	o	o	o	o	.	o	o	o	.	.	.	.	.	.	
27,18	,10 pitch,corresp,sel	o	o		.	.	.	.	.	.	.	.	.	o	.	o	o	o	o	o	.	.	.	.	.	.	.	.	.	.	
27,19	,10 pitch,stndrd,sel	o	o		.	.	.	o	.	.	.	o	.	o	o	o	J	J	J	o	.	.	.	.	.	.	.	.	.	.	
27,29	,12 pitch,corresp,sel	o	o		.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	
27,23	,12 pitch,stndrd,sel	o	o		.	.	.	.	.	.	.	.	.	o	.	o	M	o	o	o	.	.	.	.	.	.	.	.	.	.	TYPE FACE, PITCH
27,20	,condensed,select (new)	o	o		.	.	.	o	.	.	.	o	.	o	o	o	J	J	J	o	.	.	.	.	.	.	.	.	.	.	
27,14	,condensed,select (old)	o			.	.	.	.	.	o	.	.	.	.	.	.	J	J	J	.	.	.	.	.	.	.	.	.	.	.	
27,15	,condensed,cancel (old)	o			.	.	.	.	.	o	.	.	.	.	.	.	J	J	J	.	.	.	.	.	.	.	.	.	.	.	
27,66	,italics	o	o		.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
27,77	,microfont	o	o		.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
27,16, n1,n2	Position to point position	M	M	M	.	.	.	.	.	.	S	o	S	o	.	L	L	L	L	o	.	.	.	.	.	o	.	.	.	.	
27,1 - 27,6	Proportional space 1-6 pts	M	M	M	.	.	.	.	.	.	.	o	.	o	.	o	o	o	o	o	.	o	.	o	.	.	.	.	.	.	
27,1 - 27,9	space 1-9 msp				.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	HORIZONTAL POSITION
8,n	Backspace n pnts/microspces	M	M		.	.	.	o	.	.	.	o	.	o	.	o	o	o	o	o	.	.	o	o	.	.	.	.	.	.	
8	Backspace one character	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	o	.	.	o	.	.	.	.	o	
27,24	Daisy Wheel prmrtrs, start		o		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	o	o	.	.	.	.	.	.	DAISY WHEEL
27,25	, end		o		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	o	o	.	.	.	.	.	.	
27,73, n1,n2, d1,d2,d3	High-res graphics, print	P	P	P	.	.	.	.	.	.	.	.	.	K	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	HIGH RESOLUTION

CODE	FUNCTION	DP	WP	GR	1	2	3	4	5	6	7	8	1	1	1	2	4	4	5	2	D	D	2	4	1	2	1	2	T	P	
27,75	Insert new paper	o	o	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	SHEET FEED
27,76	Insert new paper	o	o	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	
28,n1,n2	Repeat print data	P	P	P	.	.	.	.	.	.	o	o	o	o	.	o	o	o	o	o	.	.	.	.	.	o	.	.	o	.	REPEAT
17	Text mode, select	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	
11	Line feed, reverse	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	CGP-115
29	Pen holder, rotate	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	
28,84,n1	Color, select color n1	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	
27,67	, scan mode	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	CGP-220
27,78	Dot pitch ratio 1:1	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	
27,80	4:3	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	
7	Audio alarm	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	
29	80 chars per line	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	
30	40 chars per line	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	
31	20 chars per line	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	
15	16 chars per line	o			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	OLDER PRINTERS
27,32	Print cents sign	P			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	
27,127	Print xxxx sign	P			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	
127	Clear buffer	P			.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	
Block Graphics Characters					.	.	.	.	o	o	.	o	.	.	o	o	o	o	o	o	.	.	.	.	.	.	.	.	.	.	
European/Japanese Character Sets					.	.	.	.	T	T	T	T	.	TUT		T	T	T	T	T	.	V	V	V	.	V	.	.	.	.	OTHER PARAMETERS
Parallel Interface					o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	.	o	
Serial Interface					.	.	.	.	.	.	Y	Z	Z	Z	Z	Z	Z	Z	.	.	.	.	Z	.	Y	1	.	Y	Y	.	

LEGEND

Data processing, graphics only
A CRLF always
B Also code 138
C Also code 141
D LF only, also code 138
E CR only
F Also code 11
G LF or CRLF switch selectable
H CR or CRLF switch selectable
I LF only
J Codes switch selectable
K Use d1, d2 only
M Motion
N Sets 10 characters per inch
P Printing
Q Sets 12 characters per inch
R CRLF except after 27,21
S Use n1 only
T European character set
U Japanese character set
V Subset of European characters
Y 600 bits per second only
Z 600/1200 bits per second
1 600/2400 bits second

SECTION 2

Printing Text

CHAPTER 7

TEXT AND WORD PROCESSING PRINTING

Up to this point we really haven't given you much in the way of solid ''code'' that you can use on your printer. In this section we'll try to remedy that situation—it's dedicated to giving you practical examples of how you can perform different printing functions. Usually the code will apply to almost all printers; but, in other cases, special techniques are used for older printers or for printers that are somewhat unique.

This section is organized functionally, from simpliest function to most complex. It's a mix of general topics and specific information. You'll find material on how to proportionally space characters along with a program to print the text screen of your system. The table of contents or index will point you to the location of techniques for the function or application in which you're interested. This chapter covers text processing only—refer to Section 3 for information on graphics.

Note: LPRINT is used as the general print command in this section and the next. If you have a Color Computer or MC-10, simply substitute PRINT#-2, (don't forget the comma) for every LPRINT you see.

LISTING PROGRAMS AND DATA

When you are in BASIC, but not running a program, you can print the material currently in memory by a simple command that works on all Radio Shack systems—LLIST. The LIST command (one L) displays the current program on the screen, and the LLIST (two Ls) is similar except that the program is sent to the system printer instead of the screen. Suppose that you had this program in memory

```
100 LPRINT "William Barden, Jr."
110 LPRINT "200 N. S. Memory Lane"
120 LPRINT "Computer City, CA 92692"
130 LPRINT: LPRINT: LPRINT
140 GOTO 100
```

It could be listed by entering an LLIST after the BASIC OK or > prompt message:

```
OK
LLIST
```

or

```
> LLIST
```

The listing would be done at the current line spacing and pitch which, in most cases, would be six-lines-per-inch spacing and 10-characters-per-inch pitch.

The LLIST command is virtually foolproof. If the printer is "ready," or ON LINE, the listing will proceed from beginning to end. To stop the listing at any time, just press < BREAK >, and the listing will stop.

If you want to list only a portion of the program, you can use a range of lines:

```
LLIST 100-300
```

for example, will list all lines from 100 through 300. Don't worry if the lines don't exist, the LLIST over a range will attempt to list any lines found within the range specified.

You can also use a minus sign to signify "all lines before" or "all lines after."

```
LLIST -9000
```

for example, lists all lines up to and through line 9000, and

```
LLIST 9000-
```

lists all lines from 9000 through the end of the program.

While running a BASIC program, of course, you'd use the LPRINT (all Radio Shack computers except the Color Computer and MC-10) and PRINT#-2, (Color Computer and MC-10). We covered those commands in some detail in Chapter 5 and won't repeat the descriptions here.

SELECTING FONTS IN DOT-MATRIX AND DAISY-WHEEL PRINTERS

Newer Radio Shack printers have a lot of different combinations of typefaces that can be used in printing. We'll describe some of the characteristics of the different type styles available here and show you how to use them.

There are two things over which you have control when selecting a type face in printers—the type style and print density.

DAISY-WHEEL PRINTERS

There's a large variety of type styles available for the daisy-wheel printers. The printing on daisy-wheel printers looks as good or better than a typewriter and approaches the appearance of copy produced by a typesetter.

Selection of a type style for daisy-wheel printers is largely a question of

which one looks best to you. There are three main categories of type wheels: those designed to be used at 10 characters per inch (10 pitch), those designed to be used at 12 characters per inch (12 pitch), and those designed to be proportionally spaced. A 10-pitch type is sometimes called "pica" and a 12-pitch type "elite." Proportional spacing type wheels are designed to be used with variable widths, rather than a fixed pitch (see "Proportional Spacing" in this section).

Since the type style is designed for either 10 pitch, 12 pitch, or proportional spacing, there's not too much variation in the "print density" in daisy-wheel printing—you choose the best type wheel for the task and stick with it.

Normally, you set the front panel switch on the daisy-wheel printer to 10, 12, or PS, based upon the type wheel mounted. However, you can select pitch through software if you wish. You can compress a short amount of text, for example, by selecting 12-pitch for a 10-pitch wheel. Selecting 10-pitch is done by sending a CHR$(27);CHR$(15) to the printer, selecting 12-pitch is done by sending a CHR$(27);CHR$(14), and selecting proportional spacing is done by sending a CHR$(27);CHR$(17). The pitch selected would remain in force until you sent a new command or powered up the printer again. Turning on the printer causes it to read the pitch setting from the front panel switches.

Here's a sample segment of code to produce a short, 12-pitch piece of text for a 10-pitch wheel.

```
100 LPRINT "This is 10 pitch";CHR$(27);CHR$(14);
110 LPRINT " this is 12 pitch";CHR$(27);CHR$(15);
120 LPRINT " and back to 10 again"
```

The result is shown in Figure 7-1.

```
This is 10 pitch, this is 12 pitch, and back to 10 again
```

Figure 7-1. Twelve Pitch Segment in Ten Pitch Mode

DOT-MATRIX PRINTERS

Type Styles There are either one or two styles of type you can choose with a dot matrix printer. Older dot-matrix printers have just one style of type—the plain vanilla dot-matrix standard type. This is a type that has no frills; it prints a simple dot matrix. Newer printers, though, are oriented more toward word processing and pretty type. Newer printers usually offer both a simple dot-matrix type, called standard and a fancier type, called correspondence. Correspondence is usually a higher-resolution type that uses a denser dot-matrix to form characters that look less like dot-matrix and more like daisy-wheel type. In the best case, the DMP-2100 uses a 24-pin print head to print characters that look almost as good as daisy-wheel characters.

Print Density There are three basic "pitches" in dot-matrix printers—normal, compressed, and condensed. Your printer may have all or some of these, although it will always have the normal pitch. Normal pitch prints at 10 characters per inch. Compressed print has 12 characters per inch. Condensed print has a density of 16.7 characters per inch. Depending upon the type of printer you have, you can select both the style, standard or correspondence, and the pitch of the type. You could print in standard style type in normal (10 cpi), compressed (12 cpi) or condensed (16.7 cpi) on the DMP-500, for example.

Compressed and condensed print densities allow you to print a lot more characters on the same line. If you have an 8-inch print line, you'll be able to get 96 characters in compressed mode and 133 characters in condensed mode, compared to the 80 characters in normal pitch. By the way, the reason for the condensed pitch of 16.7 characters per inch is that this print density can pack 132 columns in under eight inches. Since many larger printers and computer systems print on 14 7/8-inch, 132-column green-bar paper, the condensed pitch allows you to get the expanded printout on narrower width paper—a handy feature.

In addition to the three standard print densities, you can also select an elongated print mode in dot-matrix printers. This is usually a feature that you can select in addition to one of the other densities. In elongated mode (see "Elongated Printing," in this chapter), characters are printed at twice the normal width horizontally by printing twice the number of columns of dots. When elongated mode is used with normal pitch (10 cpi), the print density becomes five characters per inch. When used with compressed printing (12 cpi), the density becomes six characters per inch. When used with condensed printing (16.7), the density becomes 8.35 characters per inch.

In addition to the print densities described above, newer printers also have a proportional-spacing mode. In this mode, characters are printed at variable widths. Each printer using this mode has a table of character widths in the printer ROM, which defines the width of each character. Proportionally spaced characters look neater and are more readable than characters printed at a fixed width.

Printer Hint

DMP-110 ITALIC AND MICROFONT CHARACTERS

The DMP-110 has two unique font types not found on other printers, italic and microfont.

The italic style uses a 12-by-16 dot-matrix and is used to highlight portions of text. To set italics in either data-processing or word-processing mode, use a CHR$(27);CHR$(66). To reset italic, set any of the other print styles. Figure 7-2A shows a sample of italics.

The microfont type style is a five-by-eight dot-matrix designed for superscripting and subscripting. (The vertical height is one-half of normal, but eight dots are printed!) Set the microfont by CHR$(27);CHR$(77), and reset it by setting one of the other print styles. Figure 7-2B shows a sample of the microfont.

SELECTING TYPE STYLES AND PRINT DENSITY

Type styles and print densities in dot-matrix printers are selected by escape sequences. The escape sequence starts with a CHR$(27); and ends with a second CHR$() value. Once the type style or print density is in force, it remains in force until a new type style or print density is chosen.

Table 7-1 shows the possible combinations of type styles and print densities for every printer in this book. Table 6-1 shows which escape sequence is used to set them.

Normally you can intermix type styles at any time. You could print some text in normal pitch, immediately followed by text in condensed mode, and then reset the normal pitch again. Figure 7-2 shows a sample printing of a mixture of type, and the program used to produce the printout.

SETTING AND RESETTING DATA, WORD PROCESSING, AND GRAPHICS MODES

The LPVIII, DMP-110, DMP-200, DMP-400, DMP-420, DMP-500 and DMP-2100 all have both data-processing and word-processing mode, in addition to graphics mode.

Note: Both data-processing and word-processing modes are "text printing" modes—both modes produce text characters without graphics. The major difference in the two text modes is this: In word-processing mode some line-spacing commands are acted upon immediately, causing the

Table 7-1. Type styles and print densities

Printer	Type Styles, Print Density
LPI	Standard 10
LPII	Standard 10[1]
LPIII	Standard 10[1]
LPIV	Standard 10[1], condensed 16.7[1], proportional[1]
LPV	Standard 10[1], condensed 15[1]
LPVI	Standard 10[1], condensed 15[1]
LPVII	Standard 10[1]
LPVIII	Standard 10[1], condensed 16.7[1], proportional[1]
DMP-100	Standard 10[1]
DMP-110	Standard 10[1], elite 12[1], condensed 17[1], correspondence 10[1], correspondence elite 12[1], proportional, italic 10[1], super-, sub-, microfont 17[1]
DMP-120	Standard 10[1], condensed 16.7[1]
DMP-200	Standard 10[1], compressed 12[1], condensed 16.7[1], correspondence 10[1], proportional[1]
DMP-400	Same as DMP-200
DMP-420	Same as DMP-200
DMP-500	Same as DMP-200
DMP-2100	Standard 10[1], compressed 12[1], condensed 16.7[1], correspondence 10[1], correspondence 12[1], proportional[1]
DW-I, -II, -IIB, DWP-210, 410	Varies with type wheel
CGP-115	Standard 10[1], others
CGP-220	Standard 10[1], also dot pitch ratio 1:1
QP-I	Standard 5, 10, 20
QP-II	Standard 9[1]
TP-10	Standard 10[1]
Plotter/Printer	Standard 10, others

[1] = Halve pitch for elongated mode

This is text at 10 characters per inch,
this is condensed text, and this is normal again.

```
100 LPRINT "This is text at 10 characters per inch, ";
110 LPRINT CHR$(27);CHR$(20);"this is condensed text,";
120 LPRINT CHR$(27);CHR$(19);"and this is normal again."
```

Figure 7-2. Mixture of Type Styles

But by God, Eliot, *it was a photograph from life!*

Figure 7-2A. DMP-110 Italic Type

You can intersperse Microfont Text at any time and use it for superscripts.

Figure 7-2B. DMP-110 Microfont

paper to be moved up or down. This is primarily for superscripting and subscripting, but also applies to other line spacing. In data-processing mode, the line-spacing command is remembered and acted upon at the end of a line; it stays in force for all lines until reset.

The printer is set in data-processing mode every time the power is turned

on. To set word-processing mode, the CHR$(20) code is sent to the printer.

100 LPRINT CHR$(20)

Word-processing mode will stay in force until either a "select data-processing mode"command or "select graphics mode" command is sent. To reset data-processing mode, send a CHR$(19)

100 LPRINT CHR$(19)

While in either data- or word-processing mode (or text mode on earlier printers), graphics mode is selected by a CHR$(18).

100 LPRINT CHR$(18)

You can switch around from mode to mode at any time, even in the middle of a line, although normally you'd want to stay in either word-processing or data-processing mode for the bulk of a document. The code CHR$(30) ends graphic mode and restores whatever mode was set previously.

LPRINT CHR$(30)

Figure 7-3 shows how to print a mix of data processing and graphics—a box with text inside it. (This program was written for the DMP-2100; dot-column spacing may be different for your printer.)

Note: DMP-400, DMP-420, and DMP-500 printers have DIP-switch settings to select graphics. Normally the switches are set for the new codes to start and end graphics, (CHR$(18) and CHR$(30)). However, the "old" codes of CHR$(27);CHR$(19) (start) and CHR$(27);CHR$(20) (end) can be selected so that older applications programs which print graphics will run on the newer printers without problems. Use the old switch settings only if you are running older applications programs.

Printer Hint

FONT SELECTION BY SWITCHES

You can select the font, or type style, and print density in the DMP-2100 by setting its DIP switches. The DIP switches are under the cover, in the front right of the printer. There's a chart directly below the DIP switches. When you set the switches, the printer will read the switches every time it is powered up and then choose a type style (correspondence or standard) and a print density (10, 12, or 16.7 pitch) based on the switches. You won't have to send out the escape sequence for the font or print density in this case—the printer will use the DIP switch settings as the default value. You can still change the type style or density, however, by issuing the proper escape sequence commands to override the DIP switch settings at any time. If the bulk of your printing work is in correspondence-10 for letter writing, for example, you'd probably want to set the DIP switches for that default, and the printer would be ready to go for your word processing when you powered up.

Other printers that operate in similar fashion are the DMP-200, DMP-400, DMP-420, and DMP-500. Type styles on these printers are set by a small rotary switch under the front cover. The rotary switch can be set to the desired style and print density with a small screwdriver.

POSITIONING PRINT ON THE PAPER

One of the biggest tasks in printing is calculating the proper position of the text on paper. If you look on a piece of paper as a type of "screen" similar to the graphics screen you have on your system, then a position on the paper screen is referenced by two schemes. The first is by lines and character positions along the lines. The second is by lines and dot-columns or microspaces.

PRINTERS WITHOUT DOT COLUMNS AND VARIABLE LINE SPACING

Read this section if you have an LPI, LPII, LPIII, LPV, LPVI, DMP-120, DW-I, CGP-115, QP-I, QP-II, TP-10 or Plotter/Printer. These printers operate with a fixed number of lines per vertical inch, usually six or six and eight, and with a fixed number of character positions along the line. The number of character positions depends upon the pitch, or number of characters per inch. Different pitches can usually be selected. See Table 7-2 to see what

```
TEXT IN BOX
```

```
90 CLEAR 1000
100 LPRINT CHR$(18);CHR$(255);STRING$(120,CHR$(129));CHR$(255)
110 LPRINT CHR$(255);CHR$(30);"     TEXT IN BOX      ";CHR$(18);CHR$(255)
120 LPRINT CHR$(255);STRING$(120,CHR$(192));CHR$(255)
130 LPRINT CHR$(30)
```

Figure 7-3. Mixture of Data Processing and Graphics

line spacing and pitches are available for your printer and how to select them. Table 6-1 shows the escape sequences used to select the spacing and pitches.

Table 7-2. Line spacing and pitches

Printer	Lines/Inch Line Spacing	Pitch (Characters/Inch, Dot Columns or Microspaces/Inch)
LPI	6	10–16.5 manually variable
LPII	6	10[1]
LPIII	6,8	10[1]
LPIV	6	10[1], 16.7[1], proportional[1]
LPV	6,8	10[1]; 15[1]
LPVI	6,8,12	10[1], 15[1]
LPVII	6,9(graphics)	10[1]
LPVIII	6,8,12	10[1], 16.7[1], proportional[1]
DMP-100	6,9(graphics)	10[1], 60 (positioning)
DMP-110	6,7.5,8.6,12,120	10[1], 12[1], 17[1], 120 (positioning)
DMP-120	6,8,12,72	10[1], 16.7[1]
DMP-200	6,8,10.2,12,72	10[1], 12[1], 16.7[1], 120-200 (positioning)
DMP-400	6,8,9.8,12,36	10[1], 12[1], 16.7[1], 60-100 (positioning)
DMP-420	6,8,10,12,36	10[1], 12[1], 16.7[1], 60-100 (positioning)
DMP-500	6,8,12,36	10[1], 12[1], 16.7[1], 60-100 (positioning)
DMP-2100	6,8,7.5,8.57,120	10[1], 12[1], 16.7[1], 180 (positioning)
DW-I	3,4,6	10, 12
DW-II	6,12	10, 10, proportional, 60 (positioning)
DW-IIB	6,12	10, 12, proportional, 120 (positioning)
DW-210	6,12,48	10, 12, proportional, 120 (positioning)
DW-410	6,12,48	10, 12, proportional, 120 (positioning)
CGP-115	6	10[1]
CGP-220	6, 8	10, 80 (positioning)
QP-I	5	5, 10, 20
QP-II	5	9[1]
TP-10	6	10[1]
Plotter/Printer	6	10, others

[1] = Halve pitch for elongated mode

Positioning text in these printers is easy. A print position on the paper is referenced by lines and a column number within the line. If you're working with six lines per inch, there are 66 lines per 11 inches of standard paper. Eight lines per inch will be 88 lines per page. The number of character positions will be nominally ten per inch, and on most printers of this type can be changed to five per inch in elongated mode.

To print "This is line 20, char pos 5", you'd have:

```
100 FOR I = 1 TO 19
110 LPRINT
```

```
120 NEXT I
130 LPRINT "    This is line 20, char pos 5"
```

Notice that in the code above, four spaces precede the printable portion of the text. The spaces position the text along the line. Another way of doing this in BASIC is to use the TAB function. The TAB function jumps over a given number of character positions. You could have used

```
130 LPRINT TAB(5);"    This is line 20, char pos 5"
```

and gotten the same effect. The tab positions used in TAB correspond to any printer column and are numbered starting from 1. Note that the tab positions are not related to any physical position on the printer paper. A TAB(20) at ten characters per inch will be two inches from the left paper margin, while a TAB(20) at five characters per inch will be four inches from the left margin. The TAB value refers strictly to the number of character positions that will be tabbed, counting from the left margin (TAB = 1).

DOT-MATRIX PRINTERS WITH DOT COLUMNS AND VARIABLE LINE SPACING

All of the previous information applies to the printers in this group when the lines per inch are fixed at six or eight and plain text characters are printed without proportional spacing or graphics, so you'd best read it before continuing if you have an LPVII, LPVIII, DMP-100, -110, -200, -400, -420, -500, -2100, or CGP-220.

In addition to the standard line and character spacing described above, the printers in this group can space in fractions of lines and character positions. This comes in handy for such things as superscripting, subscripting, and graphics spacing.

CHARACTER POSITION SPACING

Dot-matrix printer lines, for printers in this category, are divided into dot columns. A single text character for a dot-matrix printer is made up of six to 20 columns of five to nine dots each (non-elongated mode), as shown in Figure 7-4. The columns are produced automatically when a character is printed in text mode. Because the printer already has built-in electronic and microprocessor logic for spacing over columns to print the text character, why not make it possible to position the print head of the printer over any column? These printers allow that little bit extra of intelligence, and it becomes a powerful feature.

Generally the number of dot columns along a line are related to the pitch and typeface of the printing. Depending upon your printer, standard pitches are 10, 12, and 16.7 characters per inch. The greater the pitch number, the more columns that can be crammed into the spacing for one character. The number of dot-columns per inch varies from about 60 to 200 per horizontal inch. The exact number for your printer is shown in Table 7-3.

The dot-column numbering begins at 0, starting at the left margin, as shown in the table. To move the print head to any dot column, you must do a dot-positioning command. The format for this command is

```
CHR$(27);CHR$(16);CHR$(MM);CHR$(LL)
```

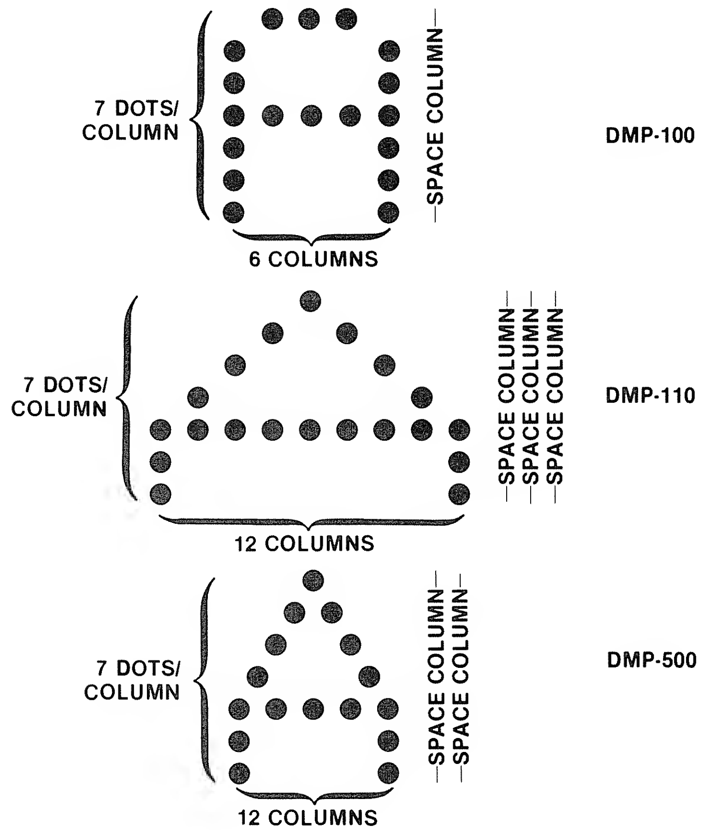


Figure 7-4. Typical Dot Arrays for Characters

The first two CHR$() characters mark the command as a column-positioning command. The next two define the dot column to the printer so it can move to the proper position. To find the MM and LL values, divide the dot column involved by 256. Save the quotient as MM and the remainder as LL. Suppose you wanted to position the print head to dot column 402, about 2.2 to 6.7 inches from the left margin, depending upon your printer and pitch. Dividing 402 by 256 gives a quotient of 1 and a remainder of 146—MM is, therefore, 1 and LL is 146. This code would print the text "TEXT" starting at dot column 402:

```
100 LPRINT CHR$(27);CHR$(16);CHR$(1);CHR$(146);"TEXT"
```

Note that a semicolon was used after the escape sequence. If a semicolon was not used, BASIC would send a carriage return after the escape sequence and cause a line feed.

Note: Because this escape sequence contains numeric values that may be any number from 0 through 255, you may have problems with the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, which is identical to the top-of-form character, the print driver will attempt to execute a page eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line-feed character, it will be changed to a 13 in the printer driver. Avoid escape sequences con-

taining these values by using a slightly larger or smaller value in the dot positioning, or use a combination of two or more positioning commands to get to the proper position.

Table 7-3. Dot columns per inch

	# Dots/inch	# Dot addressable columns	Maximum width	Graphics line space	# Dots/ vertical inch
LPI:	N/A				
LPII:	N/A				
LPIII:	N/A				
LPIV:	N/A				
LPV:	N/A				
LPVI:	N/A				
LPVII:	60 fixed	480 fixed	8″	0.11″	63
LPVIII:	120 fixed	480 fixed[5]	8″	0.1″	72
DMP-100:	60 fixed	480 fixed	8″	0.11″	63
DMP-110:	120 fixed	960 fixed	8″	0.1″	72, 144[4]
DMP-120:	120[1], 200[3]	960[1], 1600[3]	8″	0.1″	72
DMP-200:	120[1], 144[2], 200[3]	960[1], 1152[2], 1600[3] All:[5]	8″	0.1″	72
DMP-400:	60[1], 72[2], 100[3]	792[1], 950[2], 1320[3] All:[5]	13.2″	0.1″	72
DMP-420:	60[1], 72[2], 100[3]	792[1], 950[2], 1320[3] All:[5]	13.2″	0.1″	72
DMP-500:	60[1], 72[2], 100[3]	792[1], 950[2], 1320[3] All:[5]	13.2″	0.1″	72
DMP-2100:	60[1], 180[4]	816[1], 2448[4]	13.6″	0.12″	60,180[4]
DW-I	N/A				
DW-II	N/A				
DW-IIB:	N/A				
DWP-210:	N/A				
DWP-410:	N/A				
CGP-115:	N/A				
CGP-220:	80 fixed	640 fixed	8″	0.12″	60
QP-I:	N/A				
QP-II:	N/A				
TP-10:	N/A - block graphics only				
Plotter/ Printer	N/A				

N/A = No graphics
[1] = Standard [2] = Compressed
[3] = Condensed [4] = High resolution
[5] = Two dots printed for every dot column

The column-positioning command can be used at any time, whether you're in graphics or text mode. We'll show you further use of the column-positioning sequence in applications examples in this section.

Backspacing can be done on a dot column basis as can forward positioning. Use the CHR$(8);CHR$(NN) sequence for this. The NN value, however, is not a dot column number—it's the number of dot columns to space backwards. From 1 to 256 dot columns can be spaced backwards. (A value of 0 is considered 256 dot columns.) The backspace can be used for boldface printing. See "Using Bold Printing" in this chapter. Again, watch the numeric values and avoid a CHR$(12) and CHR$(10).

LINE POSITIONING

Lines, in this group of printers, are subdivided into fractions of lines. The base unit for line feeds is a *full forward line feed*, a line feed that is equivalent to six lines per inch. As we've noted before, six lines per inch is the historical line spacing that is most typical for any printer. This line spac-

ing is the default line spacing for the printer when the power is first turned on. To return to this normal spacing, a

CHR$(27);CHR$(54)

escape sequence is used.

A *half forward line feed* is found on all printers in this group and is equivalent to 12 lines per inch, making the line spacing 1/12 of an inch. One of the chief uses of half forward line feeds is for subscripting as in

H_20

The half forward line spacing is set by a

CHR$(27);CHR$(28)

escape sequence.

There's an important difference between execution of this escape sequence in data-processing mode and in word-processing mode. In data-processing mode, the line spacing is set, but not acted upon. In other words, the paper is not moved immediately after receiving the escape sequence, but is moved after a carriage return is received at the end of the line. In word-processing mode, the line spacing is *immediately* acted upon. The primary reason for this is to implement subscripting and superscripting in word-processing mode.

The half reverse line feed is the other half of the line spacing required for subscripting or superscripting. It moves the paper in reverse 1/12 of an inch (12 lines per inch). The escape sequence for this is

CHR$(27);CHR$(30)

Again, if the printer is in data-processing mode, no action is taken after the printer receives the escape sequence, until the next carriage return. In word-processing mode, though, the paper is moved immediately. Suppose that you want to print H_20 in word-processing mode. You'd do this:

```
LPRINT CHR$(20);"H";CHR$(27);CHR$(28);"2";CHR$(27);
CHR$(30);"O"
```

The CHR$(20) sets word-processing mode. The letter "H" is then printed. The CHR$(27);CHR$(28) then does a half forward line feed to space the paper 1/12 of an inch forward. The "2" is printed. The CHR$(27);CHR$(30); then spaces the paper back 1/12 of an inch to the original baseline. The "O" is then printed. The line spacing to be used at the end of the line is unchanged—it'll still be the full line feed.

If this were tried in data-processing mode, there would be no subscripting. The half forward line feed would be recognized by the printer and stored, but no paper movement would occur. The "2" would be printed on the same line as the "H" because of this. The half reverse line feed would be recognized and stored. *After the "O" was printed, the printer would be set for 1/12 of an inch reverse line feeds—not what we wanted at all!*

All of the printers in this group also have a full reverse feed, a CHR$(27);CHR$(10) escape sequence. In data-processing mode, this *sets* a full reverse line feed, but does not move the paper. In word-processing mode, the paper is moved immediately after the escape sequence is received. Here's the kicker, however—the CHR$(10) portion of the full reverse line feed is intercepted by the printer driver and changed to a CHR$(13)! To avoid

this, use two reverse half-line feeds in place of the full reverse line feed, unless you are communicating with the printer directly.

To summarize what we have covered so far:

- Full forward line feed (CHR$(27);CHR$(54)), is recognized in data-processing mode.
- Full reverse line feed (CHR$(27);CHR$(10)), moves the paper in word-processing mode but not in data-processing mode; it becomes the new line spacing in data-processing mode only.
- Half forward line feed (CHR$(27);CHR$(28)) moves the paper in word-processing mode but not in data-processing mode; it becomes the new line spacing in data-processing mode only.
- Half reverse line feed (CHR$(27);CHR$(30)) moves the paper in word-processing mode but not in data-processing mode; it becomes the new line spacing in data-processing mode only.

The remaining line-positioning commands are all *forward* commands that are dependent upon the specific printer you have. Table 6-1 shows what spacing is available, the escape sequence, and when movement occurs. Some of the spacing is used for graphics mode, to properly space the line so that no white space occurs between lines. We'll give you examples of its use in the next section.

The smallest line spacing possible in this series of printers is 1/20 forward, amounting to 1/120 of an inch. The escape sequence here (CHR$(27);CHR$(49)) is immediately acted upon in all modes. To show you what spacings are involved, try the following program. It spaces in increments of 1/120 up to 12/120 (1/10) of an inch. The result is shown in Figure 7-5.

```
100 FOR I = 1 TO 12
110 LPRINT "SPACING";
120 FOR J = 1 TO I
130 LPRINT CHR$(27);CHR$(49);
140 NEXT J
150 NEXT I
```

```
SPACINGSPACINGSPACINGSPACINGSPACINGSPACINGSPACINGSPACINGSPACING
```

Figure 7-5. Dot Matrix Variable Line Spacing

Note that the 1/120-inch line spacing is acted upon immediately and does not affect the line spacing previously set. It's used on a temporary basis to get the fine spacing required for graphics and some text.

DAISY-WHEEL PRINTERS

All of the information under "Printers Without Dot Columns and Variable Line Spacing" applies to the DW-II, DWP-210, and DWP-410 daisy-wheel

printers, as well, when the lines per inch are fixed at six and plain text characters are being printed without proportional spacing.

In addition to the standard line and character spacing described above, the printers in this group can space in fractions of lines and character positions. This comes in handy for such things as superscripting, subscripting, and justification (clean right margin).

CHARACTER POSITION SPACING

Daisy-wheel printers space in fractions of character positions by using a unit called a *microspace*. A microspace is 1/60 of an inch in the DW-II and DWP-410, and 1/120 of an inch in the DWP-210. (The DWP-410 can space in 1/120-inch units, but the basic microspace is 1/60 of an inch.) Microspaces are analogous to the dot columns used in dot-matrix printers. Microspace positioning is used to print boldface (by moving a microspace and reprinting the character—see "Bold Printing" in this chapter), and to proportionally space characters (see "Proportional Spacing").

Microspaces are not measured from the left margin as are dot columns in dot-matrix printers. The spacing is done a specified number of microspaces from the point at which the print head is positioned. To move the print head in small increments, you must perform a microspace-positioning command. The format for this command is

```
CHR$(27);CHR$(NN)
```

The NN value can be 1 through 6 for the DW-II, representing 1/60, 2/60 (1/30), 3/60 (1/20), 4/60 (1/15), 5/60 (1/12), or 6/60 (1/10) of an inch. The NN value is the same as the DW-II for the DWP-410, with the addition of a 1/120th space when NN is equal to 0. For the DWP-210, the NN value can be any number from 0 though 255. (However, watch for 10 and 12 values—use other ones instead because of printer driver interception.) The NN value represents 0/120 to 255/120 of an inch. The code below inserts spaces from 1/60 of an inch to 6/60ths of an inch, 1/120 to 6/120 of an inch on the DWP-210, between two "T" characters. The space inserted is in addition to the normal 1/10 of an inch spacing between the Ts. The result is shown in Figure 7-6 for the DWP-210.

```
100 FOR I = 1 TO 6
110 LPRINT "T";CHR$(27);CHR$(I);"T"
120 NEXT I
130 PRINT
```

```
TT
TT
T T
T T
T T
T  T
```

Figure 7-6. Daisy Wheel Microspacing

Note that microspacing commands don't affect the normal pitch of the characters. If the pitch has been set to 10 characters per inch (the default spacing at power up, depending upon dip-switch settings) this pitch will remain in force, as will a 12-character per inch pitch if selected.

The microspacing command simply inserts a space in addition to the one inserted for 1/10 inch, 1/12 inch, or proportional spacing. Note that a semicolon was used after the escape sequence. If a semicolon is not used, BASIC sends a carriage return after the escape sequence and causes a line feed.

Micro backspacing can be done on a microspace basis in addition. The CHR$(8);CHR$(NN) sequence is used for this in the DWP-210 and DWP-410 (the DW-I and DW-II backspace on *character position* with a CHR$(8) code). The NN value can be 0 to 9 for the DWP-410, representing a backspace of 0/60 to 9/60 of an inch. The NN value can be any value from 0 through 255 on the DWP-210 and DW-IIB, representing backspaces of 0/120 to 255/120 of an inch.

LINE POSITIONING

Lines in the daisy wheel printers are subdivided into fractions of lines. The base unit for line feeds is a full forward line feed, a line feed that is equivalent to six lines per inch. As we've noted before in this book, six lines per inch is the historical line spacing that is most typical for any printer. This line spacing is in effect continuously. Other line spacings can be done to print above the current line by a one-half reverse line feed (1/12 of an inch), or by one full reverse line feed, or to print below the current line by a one-half line feed (1/12 of an inch), or 1/8-line feed (1/48 of an inch; DWP-210, DWP-410, and DW-IIB only). Table 6-1 shows the possibilities and the escape sequences to cause the paper movement.

The half-forward and half-reverse line feeds are used primarily for subscripting and superscripting. They move the paper 1/12 of an inch (12 lines per inch). The escape sequence for this is

```
CHR$(27);CHR$(28)      (half forward)
CHR$(27);CHR$(30)      (half reverse)
```

Suppose that you want to print H_20 in word-processing mode. You'd do this:

```
LPRINT "H";CHR$(27);CHR$(28),"2";CHR$(27);
CHR$(30),"O"
```

The letter "H" is printed first. The CHR$(27);CHR$(28); then does a half forward line feed to space the paper 1/12 of an inch forward. The "2" is printed. The CHR$(27);CHR$(30); then spaces the paper back 1/12th inch to the original line. The "O" is then printed.

All of the printers in this group also have a full reverse feed, a CHR$(27);CHR$(10) escape sequence. Like the other line spacing commands in daisy-wheel printers, this one is acted upon immediately to move the paper. *However*, the CHR$(10) is changed by the BASIC printer driver to a CHR$(13)—it thinks it's a line feed. This means that you must use two half-reverse line feeds in place of the full-reverse line feed unless you are circumventing the BASIC printer driver.

To summarize daisy-wheel line spacing capabilities:

- Full forward line feed is always in force.
- Full reverse line feed (CHR$(27);CHR$(10)) moves the paper immediately after it is issued.
- Half forward line feed (CHR$(27);CHR$(28)) moves the paper immediately after it is issued.
- Half reverse line feed (CHR$(27);CHR$(30)) moves the paper immediately after it is issued.

The smallest line spacing you can get in this series of printers is 1/8-line

feed forward, amounting to 1/48 of an inch and it's available on the DWP-210, DWP-410, and DW-IIB. The escape sequence here (CHR$(27);CHR$(26)) is immediately acted upon in all modes. This command can be used to create any line spacing that is a multiple of 1/48 of an inch. To show you what these spacings look like, try the following program on your DWP-210 or DWP-410. It spaces in 1/48-inch increments up to 12/48 (1/4) of an inch. The result is shown in Figure 7-7.

```
100 FOR I=1 TO 12
110 LPRINT "SPACES";
120 FOR J=1 TO I
130 LPRINT CHR$(27);CHR$(26);
140 NEXT J
150 NEXT I
160 LPRINT
```

```
SPACESSPACESSPACES
                  SPACES
                        SPACES
                              SPACES
                                    SPACES
                                          SPACES
                                                SPACES
                                                      SPACES
```

Figure 7-7. Daisy Wheel Variable Line Spacing

Note that the 1/48-inch line spacing is acted upon immediately and does not affect the normal 1/6-line spacing. A 1/6-line space is still issued for an LPRINT or carriage return character (CHR$(13)). The 1/48-inch line spacing is used on a temporary basis to get the fine spacing required for some text.

SETTING TOP-OF-FORM AND SENDING FORM FEED

Radio Shack printers have no way of knowing the size of paper in the printer. There is no mark or other means that the printer can sense to detect a new page. However, Radio Shack printers, and other computer printers, have two command sequences available that can be set to adjust the printing for different page lengths and to advance the paper to the top-of-form. You could, for instance, print three- by-five-inch index card forms using a Radio Shack printer. Typical tractor/pin feed index cards are shown in Figure 7-8. (Friction feed could also be used without major problems.)

However, before we discuss the two commands, we'll have to warn you about when they can be used. Radio Shack software generally has a FORMS control that handles formatting of printing. On the Model IV, for example, you can set the number of lines per page, the number of lines in the print area, indentation, and other parameters by using the FORMS command in TRSDOS. All data sent to the printer through TRSDOS will be formatted according to the parameters you set up in the FORMS command.

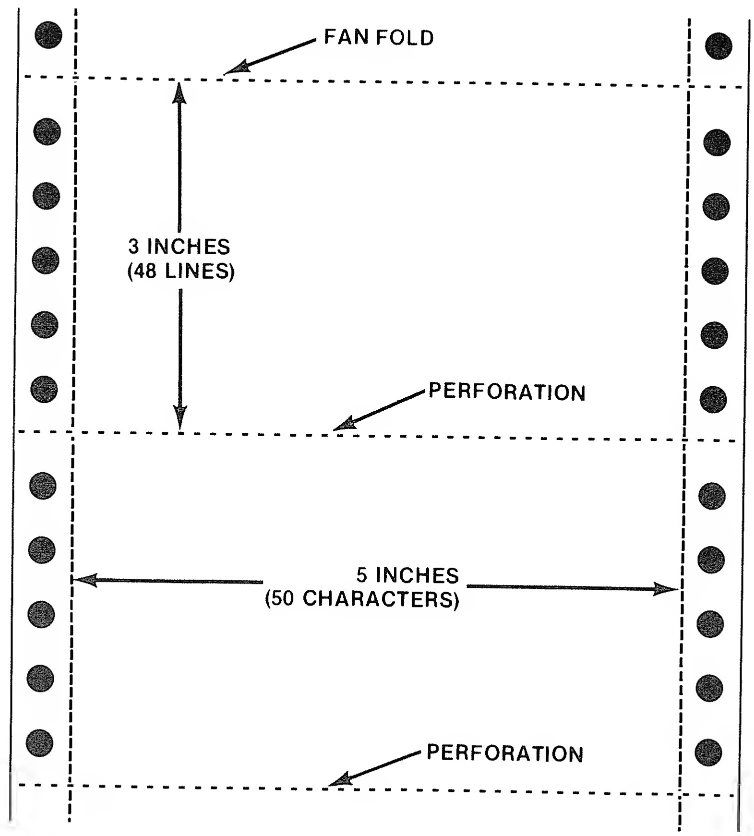


Figure 7-8. Printing Three by Five Index Cards

Printer Hint

HARD TOP-OF-FORM

Newer operating systems, such as Model IV TRSDOS and LDOS, allow you to select what's called a hard top-of-form. In this mode, the top-of-form character, a CHR$(12), is sent through the printer driver unmolested and the problem of the system intercepting the 12 code is solved—you can define your own form sizes and send a top-of-form code any time you'd like. However (there's always a catch...) if you select a hard top-of-form, some applications software that relies on a soft top-of-form may not work!

(The FORMS command is found on most other Radio Shack computers as well.) Most Radio Shack BASICs also detect (filter) the command that would cause the printer to advance to the next page and send a series of line feeds instead. In addition, most applications programs also control page formatting. Scripsit and SuperScripsit, for example, allow you to set the page length and other parameters. Although the FORMS control commands are recognized by the printer, they won't usually be received by the printer because of this filtering by the system. About the only approach left is to use your own *assembly language program*; assembly language commands usually are not filtered by the system. With that in mind, let's describe what the two commands do.

SETTING TOP-OF-FORM

The escape sequence CHR$(27);CHR$(52);CHR$(NN) sets the form length in the printer. This command *will* be sent to the printer through BASIC. The NN parameter may be a value of 0 through 255, and it represents the number of lines per page. (A 0 value is considered 256.) The form length is automatically set to 66 at power-up of the printer, 66 being the number of lines per inch for a standard 11-inch page at six lines per inch. Going back to our previous example of printing three-by-five index cards, the top-of-form command could be used to set the form length here by

```
LPRINT CHR$(27);CHR$(52);CHR$(18)
```

which would set the form length to 18 lines per page (three inches at six

lines per inch). This is a good time to add that the units here are based loosely on six-lines-per-inch line spacing, but they could just as well be based on eight lines.

TOP-OF-FORM

Having set the form length, how do you control the form spacing or page-eject? This is done through a top-of-form character—a CHR$(12). The CHR$(12) *is not* sent to the printer in most BASICs; it's usually intercepted by BASIC or the operating system software and a series of line feeds are sent to the printer instead, based on a line count and form length held *in BASIC*. However, if a CHR$(12) does get to the printer via an assembly language program, the CHR$(12) would cause this action: The printer would compare the current line count with the number of lines per form. It would then space the paper until the same position on the next page was reached, as shown in Figure 7-9. Executing a series of CHR$(12) commands would space three inches for every CHR$(12).

```
100 LPRINT CHR$(12);CHR$(12);CHR$(12)
```

would space 9 inches.

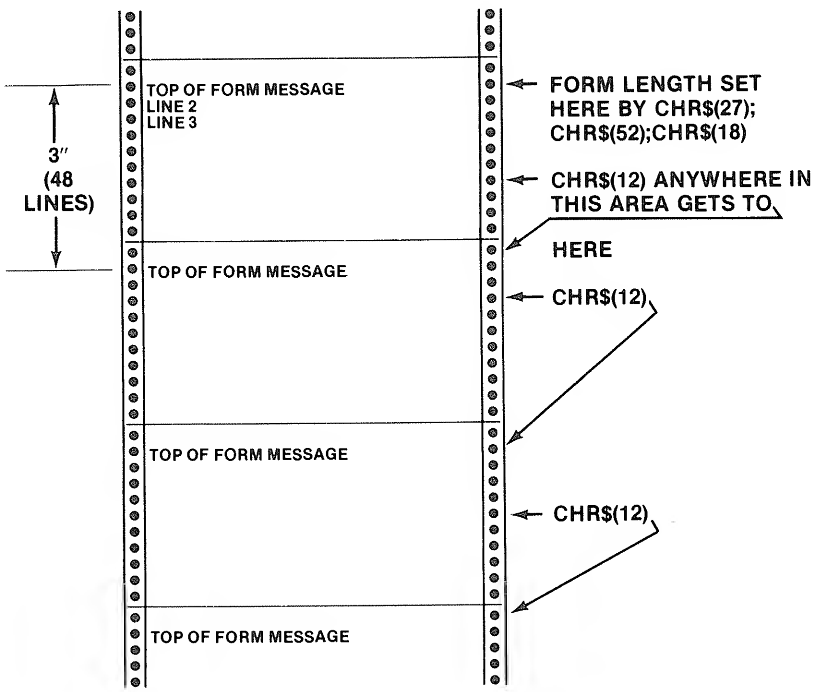


Figure 7-9. Top of Form Spacing

Listings 7-1 and 7-2 show two programs, one for the Model III and 4 and the other for the Color Computer, that show you how the top-of-form can be sent directly to the printer in assembly language. The words "TOP OF 3-INCH FORM" are printed on 10 forms, each three inches in length. They will run directly as a BASIC program; the assembly language program is coded in DATA values, relocated to RAM memory, and then executed from BASIC.

Printer Hint

WHEN IS TOP-OF-FORM SET?

Top-of-form is set automatically on power up to whatever position the paper is in. The first print line is beneath the print head, at that point. Top-of-form is also set to the current paper position after you've run out of paper and reset the system. What this amounts to is that you should position paper or forms, initially, to the first vertical line or area you want printed.

Printer Hint

HOW THE TOP-OF-FORM PROGRAMS WORK

The top-of-form programs contain a short assembly-language program in *machine language* form. The assembly language program is moved to an unused area of memory so that it can be called by the BASIC program as a subroutine. The BASIC portion of the program sets form length by the CHR$(27);CHR$(52);CHR$(18) sequence—the 18 value sets the length to 18 lines, or three inches at six lines per inch.

The BASIC program then prints out "TOP OF 3-INCH FORM" and "calls" the assembly language subroutine. The assembly language subroutine sends a code of 12 to the line printer directly, without going through the portion of the printer driver that intercepts the top-of-form code. The assembly language subroutine then returns to the BASIC program and repeats the action ten times, by the FOR...TO loop in the programs.

```
100 'MODEL III, 4 PROGRAM FOR TOP OF FORM
110 'PROTECT MEMORY AT 32511
120 DATA 33,232,55,126,230,240,254,48,32,249,62,12,211,248,201
130 FOR I=32512 TO 32526
140 READ A : POKE I,A
150 NEXT I
160 DEFUSR0=32512
170 LPRINT CHR$(27);CHR$(52);CHR$(18);
180 FOR I=1 TO 10
190 LPRINT"TOP OF 3-INCH FORM"
200 A=USR(0)
210 NEXT I
220 END
```

Listing 7-1. Model III, 4 Top of Form Program

```
100 ' COLOR COMPUTER TOP OF FORM PROGRAM
110 CLEAR 100,16127
120 DATA 134,12,173,159,160,2,57
130 FOR I=16128 TO 16134
140 READ A: POKE I,A
150 NEXT I
160 DEFUSR0=16128
170 PRINT#-2,CHR$(27);CHR$(52);CHR$(18);
180 FOR I=1 TO 10
190 PRINT#-2,"TOP OF 3-INCH FORM"
200 POKE 111,254
210 A=USR0(0)
220 NEXT I
```

Listing 7-2. Color Computer Top of Form Program

SETTING TOP-OF-FORM IN SOFTWARE

You can *simulate* top-of-form by setting up the proper variables in the BASIC print driver that controls the printing and intercepts the top-of-form command.

In the Model I, the variables are

Location 16424: Form length in lines plus one
Location 16425: Line count

Use the POKE command to POKE in the number of lines you want per page. Using our three-by-five index card example, this POKE would set the lines per page properly:

100 POKE 16424,19

From that point on, you could use a CHR$(12) to do a top-of-form—it would be intercepted by BASIC, but BASIC would use the 18 value as the lines per page. The line count variable holds the number of the current line. An example for printing the "TOP OF 3-INCH FORM" message is shown in Figure 7-10.

For the Model III and IV, the variables are

Location 16424: Form length in lines plus one
Location 16425: Line count
Location 16426: Character count
Location 16427: Character length per line-2

```
100 'MODEL I,III,4 TOP OF FORM EXAMPLE
110 POKE 16424,19
120 LPRINT "TOP OF FORM"
130 LPRINT CHR$(12);
140 GOTO 120
TOP OF FORM
```

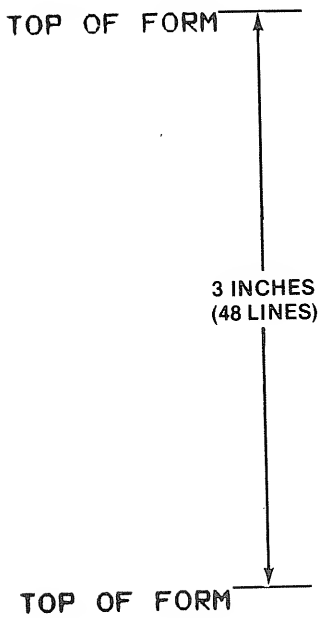


Figure 7-10. Top of Form Example

The first two variables are the same as the Model I. The variable at location 16427 controls the line length. If the line to be printed exceeds the character line length, a line feed is done automatically. This parameter is set at 255, initially, in the Model III. If you have a Model III, try this to see how the line length parameter works:

```
100 LPRINT "123456789012345678901234567890"
110 POKE 16427,25
120 LPRINT "123456789012345678901234567890"
```

You should have seen this printed:

```
123456789012345678901234567890
1234567890123456789012345678
890
```

The print driver software automatically set a line feed when the maximum number of characters per line was exceeded! The character length and line length parameters remain in force until new POKEs or a Reset occurs.

PRINTING EUROPEAN AND SPECIAL SYMBOLS

Newer Radio Shack printers have a built-in set of European symbols. Actually the term European symbols is a misnomer because there are symbols for more than just ordering Fläsk in Stockholm, seeing that your stein is fülle in Munich, or trying to eat the septième course in Paris. There are also monetary symbols (English pound and Japanese Yen), fractions (1/4, 3/4, and 1/2), and others.

Printers having this built-in set of symbols are:

LPV, VI, VIII
DMP-110, -120, -200, -400, -420, -500, -2100
DW-II (subset), DWP-210 (subset), DWP-410 (subset)
CGP-220

The complete list of symbols is shown in Table 7-4. The printers in the table don't have a completely standardized European character set, some include only a portion of the set, while others have gaps (blanks) for certain characters. The DMP-110 and CGP-220 extend the basic codes of 160-191 by adding additional codes from 192-223. The DW-II, DWP-210, and DWP-410 will not print the entire set. (The DWP-410 includes only a few characters in the set.)

The symbols are all printed by using a CHR$(NN) code in which the NN is the value of the symbol to be printed. To print the three words we mentioned (plus copyright protection for this unique code), do this:

```
100 LPRINT CHR$(171);"William Barden, Jr."
110 LPRINT "Fl";CHR$(183);"sk"
120 LPRINT "f";CHR$(184);"lle"
130 LPRINT "septi";CHR$(189);"me"
```

The result is shown in Figure 7-11

To make recalling the symbol code you used a little easier, try using a string variable with the code defined by a CHR$:

```
99 CR$ = CHR$(171): SW$ = CHR$(183): GM$ = CHR$(183):
   FR$ = CHR$(189)
100 LPRINT CR$;"William Barden, Jr."
110 LPRINT "Fl";SW$;"sk"
120 LPRINT "f";GM$;"lle"
130 LPRINT "septi";FR$;"me"
```

You can use any string variable names you'd like—ones that you find easy to remember.

Printer Hint

JAPANESE KANA SYMBOLS

Five printers—the LPVIII, DMP-200, -400, -420, and -500—have a switch selectable Japanese character set. The characters selected replace the European character set from codes 160 through 191 (in the LPVIII, the Japanese character codes go from 160 through 223). Setting a DIP switch and then powering up the printer will select either the European or Japanese character set. See Figure 3-7. The Japanese kata-kana character set is used for foreign words and phrases that have been translated into Japanese (many technical words would be in kata-kana). The symbols stand for phonetic representations of sounds and syllables, rather than objects themselves, as in *kanji* symbols, the main style of Japanese writing.

UNDEFINED CODES

Newer Radio Shack printers print a symbol that looks like an hourglass for every undefined code. The purpose of printing this symbol, instead of just ignoring the undefined code, is to let you know, via the printed copy, that you've made a mistake in an escape or control code sequence. Perfect printed copy should have no hourglass symbols for undefined codes.

Suppose that you've tried to print a CHR$(17), as in

Table 7-4. European and Special Symbols

- LPV, LPVI, DMP-120 Text
- LPVIII, DMP-200, DMP-400, DMP-420, DMP-500 Stendard, Compreseed or Condensed

Code Dec.	Code Hex	Cher.
160	A0	(Blank)
161	A1	a
162	A2	ç
163	A3	£
164	A4	(Blank)
165	A5	µ
166	A6	°
167	A7	▼
168	A8	†
169	A9	§
170	AA	(Blank)
171	AB	©
172	AC	¼
173	AD	(Blank)
174	AE	½
175	AF	¶
176	B0	¥
177	B1	Ä
178	B2	Ő
179	B3	Ü
180	B4	¢
181	B5	(Blank)
182	B6	ä
183	B7	ö
184	B8	ü
185	B9	β
186	BA	(Blank)
187	BB	e
188	BC	ù
189	BD	è
190	BE	(Blank)
191	BF	ʃ

- LPVIII, DMP-200, DMP-400, DMP-420, DMP-500 proportlonel
- DMP-110, DMP-2100, CGP-220 ell text

Code Dec.	Code Hex	Cher.	
160	A0	ˏ	
161	A1	á	
162	A2	ç	
163	A3	£	
164	A4	•	
165	A5	µ	
166	A6	°	
167	A7	▼	
168	A8	†	
169	A9	§	
170	AA	®	*
171	AB	©	*
172	AC	¼	*
173	AD	¾	*
174	AE	½	
175	AF	¶	
176	B0	¥	
177	B1	Ä	
178	B2	Ő	
179	B3	Ü	
180	B4	¢	
181	B5	~	
182	B6	ä	
183	B7	ö	
184	B8	ü	
185	B9	β	
186	BA	TM	*
187	BB	é	
188	BC	u	
189	BD	è	
190	BE		
191	BF	ʃ	

- DMP-110, CGP-220 only

Code Dec.	Code Hex	Cher.
192	C0	â
193	C1	ê
194	C2	î
195	C3	ô
196	C4	û
197	C5	^
198	C6	ë
199	C7	ï
200	C8	á
201	C9	í
202	CA	ó
203	CB	ú
204	CC	i
205	CD	ñ
206	CE	ã
207	CF	õ
208	D0	Æ
209	D1	æ
210	D2	À
211	D3	à
212	D4	Ø
213	D5	ø
214	D6	Ñ
215	D7	É
216	D8	Á
217	D9	i
218	DA	Ó
219	DB	Ú
220	DC	¿
221	DD	Ù
222	DE	È
223	DF	À

- DW-II, DWP-210, DWP-410

Code Dec.	Code Hex	Cher.	
128	80	à	
156	9C	ç	
163	A3	£	+
165	A5	µ	+
166	A6	°	
167	A7	'	
168	A8	†	+
169	A9	TM	
170	AA	®	
171	AB	©	
172	AC	¼	+
173	AD	¾	+
174	AE	½	+
175	AF	¶	+
187	BB	é	+
188	BC	ù	+
189	BD	è	+
190	BE		+
191	BF	ʃ	+
192	CO	§	+
204	CC	¥	+
219	DB	Ä	+
220	DC	Ö	+
221	DD	Ü	+
222	DE	¢	
223	DF	═	+
251	FB	ä	+
252	FC	ö	+
253	FD	ü	+
254	FE	β	+

\+ Blank on DWP-210
* Space on CGP-220

©William Barden, Jr.
Flősk
fülle
septième

Figure 7-11. Sample European Symbol Printing

100 LPRINT "HOURGLASS ";CHR$(17);"HERE"

The result you see printed is shown in Figure 7-12.

On older printers, the design philosophy was to simply ignore characters received that weren't recognized as printable characters or control codes. In newer printers, however, there are a lot more functions, each with its own escape or control code sequence, and a lot more possibility of error in sending data to a printer.

HOURGLASS ⧗ HERE

Figure 7-12. Hourglass Symbol (Elongated)

When a newer printer receives a character that it doesn't recognize, the printer generally prints an hourglass symbol, *unless* that character represents a valid control code. If the character represents a valid control code, it ignores the character and doesn't print anything. Suppose that the printer is in data-processing mode and receives a CHR$(30) character. Normally, that would be an "end graphics mode" command. The printer ignores the character. If the printer is in data-processing mode and receives a CHR$(17), though, it prints an hourglass, because the CHR$(17) is not on its list of "possible" codes.

The printer will also ignore escape code sequences in which the values are out of range. If you attempt to position the printer to dot column 65,535 by sending

```
LPRINT CHR$(27);CHR$(16);CHR$(255);CHR$(255);"TEXT"
```

the printer will ignore the positioning function. This also applies to repeat characters that are not printable characters and not possible codes—if you attempted to repeat the CHR$(17) 14 times, for example,

```
LPRINT CHR$(28);CHR$(14);CHR$(17)
```

The printer also ignores codes that are *redundant*. For example, if you tried to set graphics mode and were already in graphics mode, the code would be ignored.

To sum it all up, if you see an hourglass symbol, there's an error in your sequence of print data. Look at the point in the BASIC program that corresponds to printing commands on either side of the hourglass symbol.

ELONGATED PRINTING

Elongated printing is a cheap and easy way to get large characters on dot-matrix printers. The large characters can be used as headings or titles on reports and other documents. Figure 7-13 shows the difference between normal characters and elongated characters. The elongated characters are twice the width but the same height as normal characters. To elongate, a dot-matrix printer, in the simplest case, prints two dots for every one dot printed in the standard spacing.

```
NOT ELONGATED
N O W   E L O N G A T E D
```

Figure 7-13. Elongated Printing

The print sample in Figure 7-13 was produced by

```
100 LPRINT "NOT ELONGATED"
110 LPRINT CHR$(27);CHR$(14);"NOW ELONGATED"
```

Elongated characters can be set in any text mode in newer printers (LPII, IV, VIII, DMP-110, -120, -200, -400, -420, -500, -2100, CGP-220, TP-10) by sending a CHR$(27);CHR$(14); escape sequence to the printer. Once set, elongation will be in effect until you reset it. To reset the elongated mode, send a CHR$(27);CHR$(15).

Normal characters and elongated characters can be intermixed at any

time. Just remember that an elongated character takes up twice the width of a normal character. The twice-the-width attribute applies in whatever pitch you select. If you're in a condensed-print mode (normally 12 characters per inch), you'll get six characters per inch in the elongated character mode. If you're in a compressed print mode (normally 16.7 characters per inch), you'll get 8.35 characters per inch in the elongated mode.

Elongated characters are set in older printers (LPIII, V, VI, VII, DMP-100) by the older control code sequence CHR$(31) and reset by CHR$(32). The action, however, is the same. (Use CHR$(30) to reset elongation in the LPVII and DMP-100.) The DMP-400, -420, and -500 have dip switch settings which allow you to switch between the old and new elongated character codes. Use the new settings unless you're running software created for the older printers.

The DW-I, DW-II, DWP-210, and DWP-410 also recognize the CHR$(27);CHR$(14) and CHR$(27);CHR$(15) codes. The "elongation" here, though, is not double-width spacing, but a *more compressed* 12-character-per-inch pitch. Resetting the elongated mode restores the ten-character-per-inch normal pitch on these printers.

As with other escape codes, you can set up a string variable to use when you want to elongate in BASIC:

```
100 SE$=CHR$(27)+CHR$(14): EE$=CHR$(27)+CHR$(15)
110 LPRINT "NO ELONGATION";SE$;"ELONGATION";EE$
```

UNDERLINING

Underlining is generally set by a CHR$(15) and reset by a CHR$(14). Exceptions to this are the LPVII, in which underlining can be done by a combination of text printing and graphics. We'll show you how here.

Underlining is implemented in all "DMP" printers, all daisy-wheel printers (except the DW-I), the LPIV, LPV, LPVIII, and QP-I. The CHR$(15) and CHR$(14) can be used at any time in these printers to underline as many characters as you'd like. Here's a case in which the L, D, and A of LoaD Accumulator are underlined:

```
100 LPRINT CHR$(15);"L";CHR$(14);"oa";CHR$(15);"D";CHR$(14);
110 LPRINT " ";CHR$(15);"A";CHR$(14);"ccumulator"
```

The result is shown in Figure 7-14. Since line 100 had a semicolon on the end, there was no line feed between the "D" and the " " and the text was printed out on a single line. Note that *to avoid underlining the blank space,* underlining was turned off before printing the blank and turned on directly after.

LoaD Accumulator

Figure 7-14. Underlining Example

Underlining will remain in force until a 14 code is sent to the printer or until the printer is cycled off and on.

As for other function codes, you can use a string variable as a mnemonic to set and reset underlining:

```
100 SU$=CHR$(15): EU$=CHR$(14)
110 LPRINT SU$;"This is underlined";EU$;" and this isn't"
```

SPECIAL CASE FOR THE LPVII

Underlining can be done on the LPVII, but it's fairly involved. You must print the text, do a carriage return by a CHR$(26), switch to graphics mode by a CHR$(18), print a graphics character corresponding to the underline (CHR$(192)) over the text, and then switch back to text mode. To underline the word "SAMPLE" in the text "This is a SAMPLE", for example, you'd have:

```
100 LPRINT CHR$(30)                          'text mode
110 LPRINT "This is a SAMPLE";               'print text
120 LPRINT CHR$(26);                         'carriage return
130 LPRINT "          ";                     'space to text (10 blanks)
140 LPRINT CHR$(18);                         'set graphics mode
150 LPRINT CHR$(28);CHR$(36);CHR$(192);      'print underline
160 LPRINT CHR$(30)                          'back to text.
```

The CHR$(26) made the printer return to the beginning of the line without a line feed. The LPRINT of spaces positioned the print head over the S of SAMPLE. Graphics mode was then set by a CHR$(18). The underline character was printed by a repeat code (see "Using Repeat Codes for Text" in this chapter) that repeated the graphics character 192 a total of 36 times—36 times because there are six dots per character, horizontally in the LPVII. The 192 character prints a single dot on the bottom of the print position. Involved? Yes, but necessary for underlining on this printer.

USING REPEAT CODES FOR TEXT

Repeat codes are used when the same character must be repeated many times. It's a condensed way of instructing the printer to print the string of identical characters, and is similar to the STRING$ function in BASIC. Repeat codes are available in the LPVII, LPVIII, and all DMP printers (except the DMP-120, CGP-220, and TP-10). Suppose you wanted to print a heading on a report that looks like Figure 7-15. One way to print it is:

```
100 LPRINT "----------------------------------------------------------------"
110 LPRINT "        EFFECT OF GAMMA RAYS ON COMPUTER
MERCHANDISERS"
120 LPRINT "----------------------------------------------------------------"
```

```
----------------------------------------------------------------
        EFFECT OF GAMMA RAYS ON COMPUTER MECHANDISERS
----------------------------------------------------------------
```

Figure 7-15. Repeat Code Example

A more condensed way to print the heading uses repeat codes:

```
100 LPRINT CHR$(28);CHR$(64);"-"
110 LPRINT TAB(8);"EFFECT OF GAMMA RAYS ON COMPUTER
    MERCHANDISERS"
120 LPRINT CHR$(28);CHR$(64);"-"
```

The repeat function uses the special escape sequence CHR$(28);CHR$(NN);CHR$(CC), where NN is the number of times a

character will be repeated, and CC the value of the character to be repeated. As you can see from the examples, we can use a one-character string for the character, if the character can be entered from the keyboard. Or a CHR$() code can be used. We could have used CHR$(28);CHR$(64);CHR$(45) in the example—the CHR$(45) is equivalent to "-". The NN value can be 0 through 255. (A value greater than 255 will not be accepted by BASIC, and if sent to the printer won't be handled properly; the printer expects to see the repeat count in the next character and values greater than 255 would have to be in two bytes (characters). If a 0 value is accepted by the printer it will be treated as a 256, however.)

Note: Because this escape sequence contains numeric values that may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, identical to the top-of-form character, the print driver will attempt to execute a page eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line feed character, it will be changed to a 13 in the print driver. Avoid escape sequences containing these values by using a slightly larger or smaller value in the repeat sequence or use a combination of two or more repeat sequences to get the same result. Another good idea is to use a STRING$ function in BASIC, which operates the same as a repeat sequence, but as a BASIC and not a printer command.

The repeat function is probably most useful in graphics applications, because the printing density is so much greater than in text printing, and because it would be tedious to use a discrete CHR$() value for every graphics character to be printed. See "Repeat Codes for Graphics."

Printer Hint

BASIC STRING$ FUNCTION

The STRING$ function in BASIC is very similar to the *repeat* code in printers. The STRING$ function generates a number of identical characters as a BASIC string variable. An A$ = STRING$ (30,"A"), for example, sets string variable A$ equal to "AAAAAAAAAAAAAAAAAAA-AAAAAAAAAAA". An LPRINT STRING$ (5,CHR$(13)) sends five CHR$(13) characters to the printer (five carriage returns). The STRING$ can be used in lieu of the repeat code in printers, or in place of the repeat in those printers that do not have this repeat function. Like the printer repeat, only one character or value can be repeated in STRING$.

One reason for using the STRING$ function is to avoid problems with the BASIC printer driver, because of repeat codes that contain "problem" characters, such as 10 and 12. (The 10 is changed to a 13, and the 12 causes top-of-form action in many printer drivers.)

USING BOLD PRINTING

Bold printing can be used to emphasize titles or terms in text. It's easy to do if you have an LPV, DMP-110, -200, -400, -420, -500, or DMP-2100. It's also possible with a DW-I, DW-II, DWP-210, or DWP-410, but you have to do a little more work to get it. We'll show you how.

DOT-MATRIX PRINTERS

The escape sequence CHR$(27);CHR$(31) is used to start bold printing on the dot-matrix printers mentioned above. The escape sequence CHR$(27);CHR$(32) is normally used to end bold printing. To print the word BOLD in boldface, use this code:

```
100 LPRINT "This is ";CHR$(27);CHR$(31);"BOLD";CHR$(27);
CHR$(32);" printing."
```

Boldface characters can be printed anywhere in text printing. In bold printing, each character is printed, a slight advance along the line is made, and the character is printed again.

Most of the dot-matrix printers (all but the DMP-110, -200, and -2100) can switch to an "old" *stop bold code,* a CHR$(27);CHR$(30) in place of the CHR$(27);CHR$(32). The switch can be made by setting DIP switches.

There's a conflict between old and new half reverse line feeds, and bold end escape sequences, and the switches resolve it for older software. Use the new switch settings unless you're running older applications software, which uses the old escape sequence.

You cannot mix bold printing and elongated printing on the DMP-200, DMP-400, DMP-420, and DMP-500. You *can* mix them on the DMP-110 and the DMP-2100.

DAISY-WHEEL PRINTERS

There is no built-in command to print boldface on the daisy-wheel printers. Bold printing must be done via the software, whether it's your own BASIC program or an applications program. Printing bold characters is usually done by printing the character, backspacing to the start of the character, and overprinting the same character. (Another way of doing it is to backspace to one microspace less than the start of the character and reprint the character, but this may cause some double-image registration problems.) This technique is easy when the character pitch is constant, as in 10-character-per-inch or 12-character-per-inch spacing. However, when proportional characters are being printed, the width of each character is variable, and it's hard to define what the backspace should be in terms of microspaces.

To print bold for text that is not proportionally spaced, it's probably best to use a subroutine to print each character, because it would be tedious to include all of the escape codes after each character of text. The subroutine would look like this:

```
10000 ' PRINT BOLD
10010 ' ENTER WITH ZZ$=STRING TO BE PRINTED
10020 ' EXIT AFTER PRINTING STRING IN BOLD
10030 Z1=10: D1=0
10040 FOR I=1 TO LEN(ZZ$)
10050 LPRINT MID$(ZZ$,I,1);
10060 IF D1=1 THEN LPRINT CHR$(8); ELSE LPRINT
CHR$(8);CHR$(Z1/2);CHR$(8);CHR$(Z1/2);
10070 LPRINT MID$(ZZ$,I,1);
10080 NEXT I
10090 RETURN
```

Before you try the subroutine, do this: If you have a DW-I printer, change D1 = 0 to D1 = 1. Change Z1 to the pitch you're using — 10 or 12 characters per inch by setting Z1 to 6 for DW-I, DW-II, or DWP-410 in 10 pitch, 5 for a DW-I, DW-II, or DWP-410 in 12 pitch, 12 for a DW-IIB or DWP-210 in 10 pitch, 10 for a DW-IIB or DWP-210 in 12 pitch.

You can now call the subroutine from any point in your BASIC program to print bold. To print the word BOLD in boldface in the text "This is BOLD printing", you'd have something like this:

```
100 LPRINT "This is "
110 ZZ$="BOLD": GOSUB 10000
120 LPRINT " printing"
```

The subroutine first prints one character, using a BASIC MID$ command to access the character. It then backspaces 10 or 12 microspaces, depending upon what value you give Z1. To avoid printer driver problems from using a backspace sequence with a 10 or 12, the program backspaces in

two steps, half the distance each time. For the special case of the DW-I, a backspace alone will prevent the printer from spacing after printing. The character is then reprinted, causing a bold impression. This sequence is repeated for all of the characters in the string. After the last character has been printed twice, the subroutine returns to the main program. Read "Proportional Spacing," in this section, to see how bold printing is done for proportional spacing.

Printer Hint

MORE ABOUT BASIC SUBROUTINES

You may use as many subroutines in your BASIC programs as you'd like. Each subroutine can start with any series of commands, but must *end* with a RETURN command. Subroutines can be as few lines or as many lines as you'd like. Any time a sequence of commands or print data is repeated, it's a good chance to incorporate them into subroutine code to save repeating the commands each time. Subroutines can also call other subroutines, which can call other subroutines, and so forth. If you're unclear about how subroutines work, study some of the ones in this book, and spend an hour or so with the BASIC manual for your system—it'll be worth the trouble and save you time in BASIC print programs.

SUPERSCRIPTING AND SUBSCRIPTING

Superscripting and subscripting are done by using half line feeds, as described in "Positioning Print on the Paper." The escape sequence for half forward line feed is CHR$(27);CHR$(28) and for half reverse line feed is CHR$(27);CHR$(30). Superscripting and subscripting are available on the LPIV, LPVIII, DMP-110, -200, -400, -420, -500, -2100, DW-II, DWP-210, and DWP-410.

DAISY-WHEEL SUPER- AND SUBSCRIPTING

If you're using a daisy-wheel printer to subscript or superscript, simply insert the escape sequence for the half reverse (superscript) or half forward (subscript) directly before the text you want to subscript or superscript. After the text, you'll need to move the paper the other direction to return to the original base line. See Figure 7-16. Suppose that you wanted to print the text "A_2 is the term for I_{cc} as we know from the study of "N-dimensional Lattices in Fort Worth". We'd use these sequences

```
100 LPRINT "A";CHR$(27);CHR$(30);"2";CHR$(27);CHR$(28);
" is the term for I";CHR$(27);CHR$(28);
110 LPRINT "cc";CHR$(27);CHR$(30);" as we know..."
```

A^2 is the term for I_{cc} as we know...

Figure 7-16. Super- and Subscripting Example

A less complicated way to do this is to set the sequence for a half reverse line feed and full reverse line feed equal to a string. Let's call the half reverse string US$ for upper script and the half forward string DS$ for down script. We have this:

```
99 US$=CHR$(27)+CHR$(30): DS$=CHR$(27)+CHR$(28)
100 LPRINT "A";US$;"2";DS$;" is the term for I";DS$;
110 LPRINT "cc";US$;" as we know . . . "
```

This makes the coding much less tedious. Of course, if you're using a word processing program, such as SuperScripsit, you don't have to worry about putting in the proper codes, the program will do it for you.

After the paper moves, the old line feed spacing is retained. In other words, doing a half line feed doesn't mean that the printer is set at that spacing—subsequent line feeds will be full 1/6-inch line feeds.

DOT-MATRIX PRINTER SUPER- AND SUBSCRIPTING

If you're using a dot-matrix printer that has half forward and half reverse capability, you must be in word-processing mode to perform the subscript-

ing or superscripting. This is because the half forward and half reverse line feed codes are not acted upon immediately in data-processing mode — the line feed is set to be acted upon at the end of the current line. If you had not set word-processing mode before printing the example above, the text would have been printed as

A2 is the term for Icc as we know..."

The reason for the erroneous print is that each line feed code is remembered by the printer, but not acted upon (the paper doesn't move). The first half reverse line feed is overwritten by the next half forward line feed code, which is then overwritten by the half forward line feed code, which is then overwritten by the half reverse line feed code! At the end of the line, the line spacing remains at half reverse line feed. If subsequent lines were then printed, they'd space in reverse, at 1/12-inch spacing as shown in Figure 7-17.

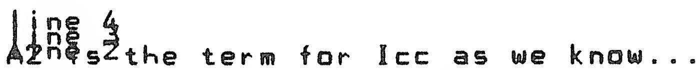


Figure 7-17. Super- and Subscripting in Data Processing Mode

Always use word-processing mode for super- or subscripting!

The LPIV, by the way, acts as if it were in word-processing mode; it was the first word-processing printer for Radio Shack.

The DMP-110 uses a one-half size *microfont* to achieve super- and subscripting.

BACKSPACING AND STRIKETHROUGHS FOR FIXED PITCH

There are times when you'd like to backspace and overtype a character or string of characters you've just printed. You might want to "slash" a zero, for example (the zero is not slashed on the DMP-2100), or you might want to display a number of strikethroughs in a contract by printing a dash or slash through text. It's easy to do this if you have an LPIV, LP VIII, DMP-110, -200, -400, -420, -500, DW-I, DW-II, DMP-210, DMP-410, CGP-115, or Plotter/Printer. These printers all have the capability to backspace one character position, and sometimes on a dot-column, or a microspace, basis. This description refers to a *fixed-pitch* backspacing. It's possible to backspace in proportional mode, but it's a little more work — see "Proportional Spacing" in this section.

BACKSPACING FOR THE DW-I, DW-II, CGP-115, OR PLOTTER/PRINTER

These printers all use a CHR$(8) character to backspace one character position. To print the text "The contract is effective on January 8,1984 February 20, 1984" with the text "January 8, 1984" slashed through, you'd have this:

```
100 LPRINT "The contract is effective on January 8, 1
984"; STRING$(14,CHR$(8));"////////////// February 20, 1984"
```

The text "The contract is effective on January 8,1984" would first be printed. At this point the print head is positioned over the next character

position after the "4" in 1984. The STRING$(14,CHR$(8)) now sends 14 CHR$(8) characters, which backspace the print head to a position over the "J" in January. Fourteen slash characters are now printed, followed by " February 20, 1984" as shown in Figure 7-18.

```
The contract is effective on J̸a̸n̸u̸a̸r̸y̸/8̸,1̸9̸8̸4̸ February 20, 1984
```

Figure 7-18. Backspacing Example

BACKSPACING FOR THE LPIV, LPVIII, DMP-110, -200, -400, -420, -500, and -2100

These dot-matrix printers all use the sequence (not an escape sequence) of CHR$(8);CHR$(NN) where the CHR$(8) marks the sequence as backspace and NN is a value of 1 through 255 (1 through 126 for the LPIV). The backspace moves the print head back NN *dot columns.* Successive backspace commands can be used to move back more than 255 dot columns.

In using this code, you must know the number of dot columns involved in printing the character in order to move back to exactly the start of the previous character position(s). This depends upon the pitch and the type style selected. Table 7-5 lists the dot columns for all of the dot-matrix printers in this category.

Table 7-5. Dot column backspacing for dot-matrix printers

Printer	# Dots/character width (except proportional)*
LPIV	Standard: 10 condensed: 9
LPVIII	Standard, condensed: 12
DMP-110	Normal standard, normal correspondence, italic: 12; elite standard, elite correspondence: 10; condensed, super-, sub-, microfont: 7
DMP-200	All:12
DMP-400	All:12
DMP-420	All:12
DMP-500	All:12
DMP-2100	Standard 10: 18, standard 12: 15; condensed: 18, correspondence 10: 36; correspondence 12: 30

*Double # dots for elongated

To perform the same strikethrough as the example on an LPVIII in ten-character per inch mode, you'd have:

```
100 LPRINT "The contract is effective on January 8, 1
984"; CHR$(8);CHR$(180);"////////////// February 20, 1984"
```

Here the number of dots per character width was 12, so 12 times 15, or 180 dot columns, were backspaced to put the print head at the beginning of "January."

If elongated mode is in force, double the number of dot columns per character-width. Proportional characters are a special case, as we mentioned—see the description of "Proportional Spacing" in this section.

Note: Because the backspace escape sequence contains numeric values that may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12,

identical to the top-of-form character, the print driver will attempt to execute a page-eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line feed character, it will be changed to a 13 in the print driver. Avoid backspacing 10 or 12 dot columns.

BACKSPACING FOR THE DWP-210, DWP-410, AND DWII-B DAISY-WHEEL PRINTERS

These daisy-wheel printers all use the sequence (not an escape sequence) of CHR$(8);CHR$(NN) where the CHR$(8) marks the sequence as backspace, and NN is a value of 0 through 255 for the DWP-210, and 0 through 9 for the DWP-410. The backspace moves the print head back NN *microspaces* or 1/120 of an inch for the DWP-210, or 1/60 of an inch for the DWP-410. Use a repetitive number of these backspace commands to move back a large number of character positions.

In using this code, you must know the number of microspaces used in the character just printed. If 10 pitch is being used, this will be 12/120 on the DWP-210 and 6/60 for the DWP-410. If 12 pitch is in use, it's 10/120 for the DWP-210 and 5/60 for the DWP-410. Proportional spacing is a special case discussed under "Proportional Spacing" in Chapter 8.

To perform the same strikethrough as for example, on a DWP-210 in ten-character-per-inch mode, use

```
100 LPRINT "The contract is effective on January 8, 1
984"; CHR$(8);CHR$(90);"/////////////// February 20, 1984"
```

Here the number of microspaces per character width was six, so six times 15, or 90 microspaces, were backspaced to put the print head at the beginning of "January."

Note: Because the backspace escape sequence contains numeric values that may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, identical to the top-of-form character, the print driver will attempt to execute a page-eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line-feed character, it will be changed to a 13 in the print driver. Avoid backspace escape sequences containing values of 10 or 12.

A GENERALIZED BACKSPACE SUBROUTINE

Backspacing a large number of dot columns or microspaces is difficult because a number of individual backspaces must be done. The BASIC code below is a generalized subroutine to backspace any number of microspaces, or dot columns, up to the limit of the line. In the code below, variable ZZ specifies the number of backspaces to be done, for either a dot-matrix or daisy-wheel printer, using the CHR$(8);CHR$(NN) command. Set ZZ equal to either dot columns or microspaces, from 1 to the limit of the line length.

```
10000 ' BACKSPACE SUBROUTINE
10010 ZA=ZZ-INT(ZZ/9)*9: ZB=INT(ZZ/9)
10020 IF ZB=0 THEN 10040
10030 FOR ZI=1 TO ZB: LPRINT CHR$(8);CHR$(9);: NEXT ZI
10040 IF ZA <> 0 THEN LPRINT CHR$(8);CHR$(ZA);
10050 RETURN
```

To backspace 35 characters in 12-pitch daisy-wheel printing, you'd call the subroutine this way:

```
100 ZZ=35*12: GOSUB 10000
```

Use six in place of 12 for a DW-II. To backspace 35 characters in dot-matrix printing with 12 dot columns per character, you'd call the subroutine by:

```
100 ZZ=35*12: GOSUB 10000
```

CHAPTER 8

WORD-WRAP, JUSTIFICATION, AND PROPORTIONAL SPACING

In this chapter we'll describe how to print text lines without splitting words in the process as both justified and unjustified lines. This material applies to all dot-matrix and daisy-wheel printers with a fixed pitch — 10, 12, or 16.7 characters per inch. In another discussion in this chapter we'll describe how to do the same thing with proportional spacing.

WRAP AROUND

What happens when the number of characters to be printed on one printer line exceeds the number that can physically be printed? Suppose, for example, that you have an 80-character-per-line printer set at ten characters per inch, and then try to print these lines:

"Computer analysts were staggered today when Tandy
Corporation announced a new printer product — the
Tandy LP-XXXIV.5, a unit capable of illuminating
manuscripts. Analysts predicted reductions in
monastary work forces."

The result will print as shown in Figure 8-1. The first 80 characters, from "Computer" through the "r" in "printer" are printed on the first line. The next line starts with "inter product...". Bringing down the line in such a fashion is called *wrap around*. The alternative to wrap around is to keep on printing at the end of a line — the characters in the last print position will be overprinted until a line feed occurs. (Early teletypewriters did, in fact, do exactly that. You'd come back to a listing after coffee to find the last 1,287 characters printed in the last character position of the line, because you left out a carriage return, line feed!)

80 CHARACTERS

```
Computer Analysts were staggered today when Tandy Corporation announced a new pr
inter product - the Tandy LP-XXXIV.5, a unit capable of illuminating manuscripts
.  Analysts predicted reductions in monastery work forces.
```

Figure 8-1. Wrap Around in Printers

WORD WRAP

If you're printing text, however, you'd like to insert a line feed not at the last character of the line, as the wrap around feature does, but after a *word break*. The text should read as shown in Figure 8-2. Printers don't have enough intelligence to do this currently; although it's certainly possible to implement such a feature in printer firmware, the software contained in a printer. For the time being, we've got to implement such a feature in computer software—it's called word wrap.

LINES END AT "WORD BREAK"

```
Computer Analysts were staggered today when Tandy Corporation announced a
new printer product - the Tandy LP-XXXIV.5, a unit capable of illuminating
manuscripts.  Analysts predicted reductions in monastery work forces.
```

Figure 8-2. Word Break

How do you do this? One way involves processing a line in fixed pitch—10 characters or 12 characters per inch, horizontal spacing. Another way involves proportionally spaced characters. We'll cover the method for fixed pitch here. The method for proportional spacing in both dot-matrix and daisy-wheel printers is described in "Proportional Spacing" in this chapter.

JUSTIFICATION VS. UNJUSTIFIED LINES

When the word wrap is done on every line, the lines will have a ragged right margin. This type of margin is called *unjustified* and is fine for manuscripts and other informal printed material. Justified text, on the other hand, has a clean, straight right edge in a clean vertical column. See Figure 8-3.

It's fairly easy to produce unjustifed text. We'll use the text above to illustrate. See Figure 8-4. The process goes like this:

1. Find the number of character positions in a line, based upon the left and right margins. In this case, it's 55 character positions.
2. From text to be printed, start counting characters by word, including spaces.
3. When the total number of characters in the current line of words exceeds the number of characters in a line (at the second "r"

"CLEAN"
RIGHT EDGE

```
Computer Analysts were staggered  today when Tandy Corporation announced a
new printer product - the Tandy LP-XXXIV.5, a unit capable of illuminating
manuscripts.  Analysts predicted reductions in monastery work forces.  The
LP-XXXIV.5  is the newest model in a long line of  Roman Numeral printers.
The  first Roman Numeral  printer, the LP-I, was introduced in  MCMLXXVII,
followed by others at the rate of IV per year.
```

Figure 8-3. Justified Text

character in the word "Corporation"), go back to the end of the previous word (the end of "Tandy").

4. Print all words from the first through the last complete word on the line.

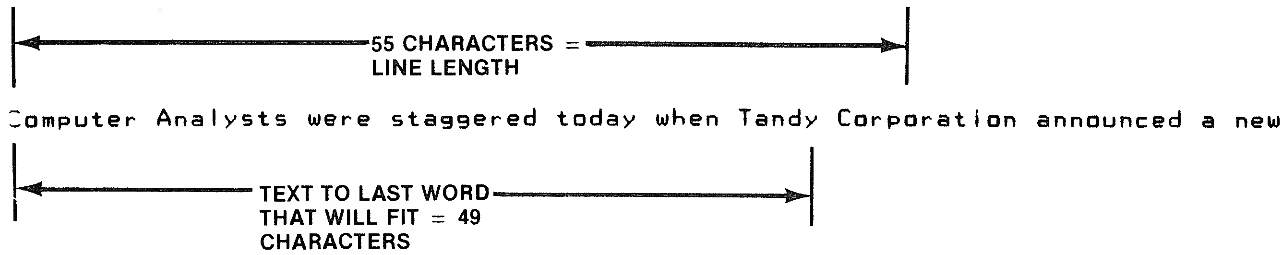


Figure 8-4. Producing Unjustified Text

A subroutine to accomplish this is shown in Listing 8-1. It will work for both dot-matrix printers and daisy-wheel printers. The subroutine will take any string of up to 150 characters or so (assumed to be words), and do a word wrap, printing the line with an unjustified right edge.

```
10000 ' FIXED PITCH WORD WRAP SUBROUTINE
10010 ' ENTRY: ZZ$=STRING
10020 '        ZR=RIGHT MARGIN IN CHARACTER POSITIONS FROM EDGE
10030 '        ZL=LEFT MARGIN IN CHARACTER POSITIONS FROM EDGE
10040 '        ZV=LINE SPACING 1=SINGLE, 2=DOUBLE, ETC.
10050 ' EXIT:  LINE PRINTED AND ZZ$=REMAINDER OF STRING
10060 ZC=ZR-ZL : IF LEN(ZZ$)<=ZC THEN ZC=LEN(ZZ$) : GOTO 10100
10070 IF ZC=0 THEN RETURN ELSE IF MID$(ZZ$,ZC,1)<>" " THEN ZF=1 ELSE ZF=0
10080 IF MID$(ZZ$,ZC,1)<>" " THEN F=1
10090 IF MID$(ZZ$,ZC,1)<>" " OR (MID$(ZZ$,ZC,1)=" " AND F=0) THEN ZC=ZC-1 :
 IF ZC>0 THEN 10080
10100 LPRINT TAB(ZL); LEFT$(ZZ$,ZC)+STRING$(ZV-1,13)
10110 ZZ$=MID$(ZZ$,ZC+1) : RETURN
```

Listing 8-1. Word Wrap Program

To use the subroutine, put the text string you want word-wrapped and printed, into string ZZ$, the left margin in character positions in ZL, the right margin character positions in ZR, and the line spacing in ZV (ZV = 1 doesn't skip lines, ZV = 2 double spaces, and so forth). Then call the subroutine and the line will be printed with word wrap. The remainder of the text will be returned in ZZ$. Suppose you had ten long strings of text to print, with a left

margin of 10, a right margin of 65, and single line spacing. The strings are in AA$(1) through AA$(10). This code would print the text with word wrap:

```
100 ZZ = "": ZL = 10: ZR = 65: ZV = 1
110 FOR I = 1 TO 10
120 ZZ$ = ZZ$ + AA$(I): GOSUB 10000
130 NEXT I
140 IF ZZ$ < > "" THEN GOSUB 10000 ELSE 160
150 GOTO 140
160 ...
```

What about printing justified lines? In this case the process is similar. You'd follow the same steps as outlined for the unjustified word wrap example. After finding the last word to fit in the line, however, you'd then do this:

1. Subtract the number of characters through the last word (49 through "Tandy") from the total number of character positions in the line (55 with ZL = 10, ZR = 65). In the previous example (Figure 8-4), it's six character positions.
2. The result of the subtraction is the number of character positions left over. This number of character positions must be spread out between words on a somewhat orderly basis, to avoid large gaps between any two words. We'll do it randomly here, putting six character positions in six interword spaces.
3. The result is a line of justified text. The text "Corporation announced a new..." is left over and applies to the justification procedure for the next line.

The subroutine shown in Listing 8-2 is similar to the previous subroutine, except that it also justifies lines. Use it with the same variables as in the previous case.

```
10000 'FIXED PITCH WORD WRAP AND JUSTIFICATION
10010 'ENTRY: ZZ$=STRING
10020 '       ZR,ZL=RIGHT, LEFT MARGINS IN CHAR POSITIONS
10030 '       ZV=LINE SPACING, 1=SINGLE, 2=DOUBLE,...
10040 'EXIT:  LINE PRINTED AND ZZ$=REMAINDER OF STRING
10050 ZB=ZR-ZL : ZC=0 : ZS=0
10060 FOR ZI=1 TO LEN(ZZ$)
10070 IF MID$(ZZ$,ZI,1)=" " THEN ZE=ZI : ZC=0 : ZS=ZS+1 : GOTO 10090
10080 ZC=ZC+1
10090 ZB=ZB-1
10100 IF ZB=0 THEN 10130
10110 NEXT ZI
10120 LPRINT TAB(ZL-1);ZZ$ : ZZ$="" : GOTO 10250
10130 IF ZS>0 THEN ZS=ZS-1 : ZI=ZE : ZB=ZC+1 : ZY$=MID$(ZZ$,ZI+1)
10140 ZZ$=LEFT$(ZZ$,ZI-1)
10150 ZF=ZB-INT(ZB/ZS)*ZS : ZB=INT(ZB/ZS)
10160 LPRINT TAB(ZL-1);
10170 FOR ZI=1 TO LEN(ZZ$)
10180 ZD$=MID$(ZZ$,ZI,1) : LPRINT ZD$;
10190 IF ZD$<>" " THEN 10220
10200 LPRINT STRING$(ZB," "); : ZS=ZS+1
10210 IF ZF>0 THEN LPRINT" "; : ZF=ZF-1 : IF ZF>ZS AND RND(2)=2 THEN 10210
10220 NEXT ZI
10230 LPRINT STRING$(ZV,13);
10240 ZZ$=ZY$
10250 RETURN
```

Listing 8-2. Word Wrap/Justification Program

PROPORTIONAL SPACING

Compare the two lines of type in Figure 8-5. Which looks better? The first line of type is similar to type produced by many typewriters. Like many computer printers, this type has a uniform width for each character—10 characters per inch in pica style and 12 characters per inch in elite style. The second line of type is produced by a typesetter. You'll notice that there are more characters in the same space and that each character has a different width. An "i," for example, occupies a much smaller width than an "M." Proportionally spaced type, like the second example, is more pleasing to the eye, and more easily scanned. It is also faster to read and less tiring.

Menus are used not only in posh restaurants, but in posh com-

Menus are used not only in posh restaurants, but in posh com-

Figure 8-5. Proportionally-Spaced Type Vs. Monospaced Type

Virtually all books and magazines and a lot of commercially prepared printing is done in proportionally spaced characters. It's logical to add proportionally spaced type to printers, and many of the Radio Shack printers have that capability. Proportionally spaced characters created by a dot-matrix printer will not look as aesthetically pleasing as those produced by a typesetter. However, daisy-wheel characters that are proportionally spaced, and the high-density characters produced by the DMP-2100, are a lot more pleasing to the eye than earlier printers which used a fixed 10 or 12 pitch and a small matrix of dots.

DOT-MATRIX PROPORTIONAL SPACING

Proportionally spaced characters are available on the LPIV, VIII, DMP-110, -200, -400, -420, -500, and -2100. Proportional spacing is selected by the escape sequence CHR$(27);CHR$(17). Proportionally spaced characters can be used in both data- and word-processing modes. Of course, when you use proportionally spaced characters, a pitch selection of 10 or 12 characters per inch is meaningless, as all characters are spaced according to their width.

To see what proportional spacing looks like on your printer, compared to normal printing, run the following short program:

```
100 LPRINT "This is not Proportional Spacing"
110 LPRINT CHR$(27);CHR$(17)
120 LPRINT "This is now Proportional Spacing"
```

The result will look like Figure 8-6.

This is not Proportional Spacing

This is now proportional Spacing

Figure 8-6. Printer Proportional Spacing

CHARACTER WIDTHS

Much of the time when you're using proportionally spaced characters, you'll be working within a word-processing program, such as SuperScripsit, which automatically handles the tasks of figuring out character widths and justification. However, it's possible to work with proportionally spaced characters in your own program. To do this, you need to know the widths for each character. These widths are found in the back of your printer manual. We won't reproduce them here as they vary quite a bit from printer to printer.

The character widths are defined in terms of dot columns. (If you're unfamiliar with how dot positioning is done, this might be a good time to read "Positioning Print on Paper," which describes dot-column coordinates.) From the table in your manual, you can see that character widths range from ten, for such characters as a period or comma, to 20, for such characters as a percent sign and uppercase W.

LINE JUSTIFICATION

There's no problem at all in using proportional-spaced characters with a ragged-right margin, as shown in Figure 8-7. Just print a line of text using an average character width of about 15 dots per character. Occasionally, lines will be longer than you desire, if there's an overabundance of wide characters, or smaller than normal if there are more narrow-width characters. In general, though, you can simply include the text you want printed in common BASIC LPRINT lines.

Printer Hint

WIDTHS FOR PROPORTIONAL SPACING CHARACTERS

You can easily verify the width data found in the back of your daisy-wheel or dot-matrix printer manual. Just set the PS switch on your daisy-wheel printer, or do a CHR$(27);CHR$(17) on your dot-matrix printer, and type out 50 identical characters. Measure the width of the resulting line, divide by 50 to get the width of an individual character, and then divide that figure again by 1/60 to get the width of the character in 1/60 of an inch units. Example: If 50 upper case H's take 5 inches, then each H is 5 inches/50 wide, or 1/10-inch. Dividing 1/10 by 1/60 gives us 5 units, the width for an H. If you type the same number of characters for all characters in a set, you can see easily which characters are the same width—they'll end up at the same point.

Menus are used not only in posh restaurants, but in posh computer
software. You've seen menus on Radio Shack software, but let's
illustrate the use of them to jog your memory. Suppose that we have
written an applications program to process weather data. When
the program is first loaded, it may display a menu of functions
that may be selected, as shown in figure 4-6. If entry 4 is desired,
then the user types a '4', and a new menu of items related to
'ANNUAL WEATHER DATA' is displayed for further selection. This
type of implementation is termed 'MENU-DRIVEN.' Menus provide an
easy-to-use format that is very descriptive. This section should def-
initely be interpreted as a plus for menu use. (I have a brother-in-
law in the menu-printing business.) Menu printing is easy, of

Figure 8-7. Dot-Matrix Proportionally-Spaced Type with Unjustified Edge

---

However, suppose that you want to justify the text shown in Figure 8-8 below. We'll describe an *algorithm*, which is essentially a plan of approach, that does the task.

1. Find the number of dot columns in a line based upon the position of the left and right margin. In this case it's 800 dot columns. (We're using a "generic" printer here, and the number of dot columns per inch may not match your printer.)
2. From text to be printed, start counting character widths by word, including spaces.
3. When the total number of dot columns in the current line of words exceeds the number of dot columns in a line (in the word "Market"), go back to the beginning of the previous word ("Computer").

UNJUSTIFIED

'The microcomputer boom is over,' analysts at Computer Market Prognostication and Rumor Co., Inc. announced today. 'There are currently over 300 manufacturers of microcomputers all claiming to be shipping units at the rate of 10,000 units per week. There are 40 million households in the U.S. and we've been checking every other one. We found micros in only 3 out of 1757 households. Clearly someone is distorting the facts.'

JUSTIFIED

'The microcomputer boom is over,' analysts at Computer Market Prognostication and Rumor Co., Inc. announced today. 'There are currently over 300 manufacturers of microcomputers all claiming to be shipping units at the rate of 10,000 units per week. There are 40 million households in the U.S. and we've been checking every other one. We found micros in only 3 out of 1757 households. Clearly someone is distorting the facts.'

Figure 8-8. Text for Justification 1

4. Find the number of dot columns in all words through the last complete word.
5. Subtract the number of dot columns in all words from the total number of dot columns per line.
6. The result of the subtraction is the number of dot columns left over. This number of dot columns must be spread out between words on a somewhat orderly basis to avoid large gaps between any two words.
7. The result is a line of justified text. The text "Market Prognostication . . . " is left over and applies to the next line's justification procedure.

You can see that it's not an easy task to justify text with proportional spacing. To simplify things for your own programming, use the subroutine shown in Listing 8-3. It will take any string of up to 150 characters (assumed to be words), and justify it, based on the total number of dot columns per line and character widths.

To use the subroutine, include the DATA statements for the character widths of your own printer. The ones we've included here are the first 96 values for the DMP-400, so if you have that printer you can leave the DATA lines as they are. Otherwise, enter a total of 160 values, one for each character from 32 through 127, zeroes for the "unused" characters from 128 through 159, and one each for values 160 through 191 for the European character set (see "Using European Symbols"). If your printer does not have code for some of the European characters or other characters, enter a zero for the width. The widths must be in strict order as shown, otherwise some characters will be the wrong width.

To use the subroutine, put the text string you want justified in ZZ$, the left margin in dot columns in ZL, the right margin in dot columns in ZR, and the line spacing in ZV (ZV = 1 doesn't skip lines, ZV = 2 double spaces, and so

```
100 CLEAR 1000
110 DIM CH(160)
120 ' WIDTH VALUES FOR DMP-400. SUBSTITUTE VALUES FOR YOUR PRINTER.
130 DATA 10,10,12,20,18,20,18,10,10,10,16,16,10,16,10,16
140 DATA 16,16,16,16,16,16,16,16,16,16,10,10,14,16,14,16
150 DATA 18,20,18,18,20,18,18,20,20,12,18,18,18,20,18,20
160 DATA 18,18,18,18,20,18,18,20,18,18,16,14,16,14,18,20
170 DATA 18,18,16,16,16,16,14,16,16,10,10,16,10,20,16,16
180 DATA 16,16,14,16,14,16,16,20,16,16,14,14,10,14,16,18,0
190 FOR I=0 TO 95
200 READ CH(I)
210 NEXT I
220 ZZ$="Your text here."
230 ZL=10: ZR=820: ZV=1
240 GOSUB 10000: IF ZZ$<>"" THEN 240 ELSE STOP
10000 ' PROPORTIONAL JUSTIFICATIONS SUBROUTINE FOR DMP-200, -400, ETC.
10010 ' ENTRY: ZZ$=STRING
10020 '        CH()=CHARACTER WIDTH ARRAY
10030 '        ZR=RIGHT MARGIN IN DOT COLUMNS FROM EDGE
10040 '        ZL=LEFT MARGIN IN DOT COLUMNS FROM EDGE
10050 '        ZV=LINE SPACING 1=SINGLE, 2=DOUBLE, ETC.
10060 ' EXIT:  LINE PRINTED AND ZZ$=REMAINDER OF STRING
10070 ZU=1
10080 LPRINT CHR$(27);CHR$(17);
10090 ZB=ZR-ZL: ZC=0: ZS=0
10100 FOR ZI=1 TO LEN(ZZ$)
10110 ZD$=MID$(ZZ$,ZI,1): IF ZD$=" " THEN ZE=ZI: ZC=0: ZS=ZS+1: GOTO 10130
10120 ZC=ZC+CH(ASC(ZD$)-32)
10130 ZB=ZB-CH(ASC(ZD$)-32)
10140 IF ZB<=0 THEN 10180
10150 NEXT ZI
10160 IF ZL=1 THEN 10170 ELSE FOR ZI=1 TO ZL-1:LPRINT CHR$(27);CHR$(ZU);: NEXT ZI
10170 LPRINT ZZ$: ZZ$="": GOTO 10310
10180 IF ZS=0 THEN 10310
10190 ZI=ZE: ZB=ZB+ZC+9+ZU: ZY$=RIGHT$(ZZ$,LEN(ZZ$)-ZI)
10200 ZZ$=LEFT$(ZZ$,ZI-1)
10210 ZS=ZS-1:ZF=ZB-INT(ZB/ZS)*ZS: ZB=INT(ZB/ZS) : PRINT ZF,ZB
10220 IF ZL=1 THEN 10240
10230 FOR ZI=1 TO ZL-1: LPRINT CHR$(27);CHR$(ZU);: NEXT ZI
10240 FOR ZI=1 TO LEN(ZZ$)
10250 ZD$=MID$(ZZ$,ZI,1): IF ZD$<>" " THEN LPRINT ZD$;: GOTO 10280
10260 FOR ZJ=1 TO ZB+9+ZU: LPRINT CHR$(27);CHR$(ZU);: NEXT ZJ
10270 IF ZF<>0 THEN LPRINT CHR$(27);CHR$(ZU);:ZF=ZF-1
10280 NEXT ZI
10290 LPRINT STRING$(ZV,13);
10300 ZZ$=ZY$
10310 RETURN
```

Listing 8-3. Dot-Matrix Proportional Justification Program

forth). Then call the subroutine and the line will be printed, justified on the right with word wrap. The remainder of the text will be returned in ZZ$. Suppose that you had ten long strings of text to print, with a left margin of 100 dot columns, a right margin of 700 dot columns, and double line spacing. The strings are in a string array of AA$(1) through AA$(10). This code would print the double-spaced text:

```
100 ZZ = "": ZL = 100: ZR = 900: ZV = 2
110 FOR I = 1 TO 10
120 ZZ$ = ZZ$ + AA$(1): GOSUB 10000
```

```
130 NEXT I
140 IF ZZ$< >"" THEN GOSUB 10000 ELSE 160
150 GOTO 140
160 . . .
```

GENERAL BACKSPACE SUBROUTINE FOR PROPORTIONAL PRINTING

Sometimes you want to backspace proportional characters for overprinting. (A good example is slashing zeroes on the DMP-2100. Unfortunately, the zero, the alphabetic O, and the D look very similar—especially if you're debugging a program at 2:00 AM). The subroutine in Listing 8-4 will allow you to backspace so you can slash zeroes or strike through other characters.

```
10000 'SUB STRIKE-OVER FOR DOT-MATRIX PRINTERS
10010 'ZZ$ = CHAR TO BE STRUCK OVER
10020 'ZY$ = CHAR TO STRIKE OVER
10030 ZU=1
10040 ZB=CH(ASC(ZZ$)-32) : FOR ZI=1 TO ZB : LPRINT CHR$(8);CHR$(ZU); : NEXT
 ZI : LPRINT ZY$;
10050 ZD=ZB-CH(ASC(ZY$)-32) : IF ZD=0 THEN RETURN
10060 IF ZD>0 THEN FOR ZI=1 TO ZD : LPRINT CHR$(27);CHR$(ZU); : NEXT ZI : R
ETURN
10070 ZD=-ZD : FOR ZI=1 TO ZD : LPRINT CHR$(8);CHR$(ZU); : NEXT ZI : RETURN
```

Listing 8-4. Dot-Matrix Proportional Backspace Program

Enter the subroutine with the character for the backspace in ZZ$ and the character to be overprinted in ZY$. At the return from the subroutine, the print head will be positioned as if the first character alone had been printed. (The second character is assumed to be equal to or smaller than the first.) The width values for the characters must be in the CH array as before.

DAISY-WHEEL PROPORTIONAL SPACING

Proportionally spaced characters are available on the DW-II, DWP-210, and DWP-410. You should have a proportional daisy-wheel in your printer for best results. (You must also set the 10/12/PS front-panel switch on your printer to the "PS" position.) The proportional character sets are designed for best appearance when doing proportional spacing. (Typewriter print faces are designed for best appearance when doing uniform spacing.) Proportional spacing is selected by the escape sequence CHR$(27);CHR$(17). When you're using proportionally spaced characters, a pitch selection of 10 or 12 characters per inch is meaningless, as all characters will be spaced according to their width.

To see what proportional spacing looks like on your daisy wheel, compared to normal printing, run the following short program:

```
100 LPRINT "This is not Proportional Spacing"
110 LPRINT CHR$(27);CHR$(17)
120 LPRINT "This is now Proportional Spacing"
```

CHARACTER WIDTHS

Much of the time when you're using proportionally spaced characters, you'll be working within a word-processing program such as SuperScripsit,

which automatically handles the tasks of figuring out character widths and justification. However, it's possible to work with proportionally spaced characters in your own program. To do this, you need to know the widths for each character. These widths are found in the back of your printer manual, but we'll reproduce them here in Table 8-1. See the cautionary note in "Printer Hints."

Table 8-1. Proportional spacing widths

(Space)	6	@	7 (6)	´^	5	
!	3	A	7 (6)	a	5	
"	4	B	6 (5)	b	5	
#	6	C	7 (6)	c	5	
$	5	D	6 (6)	d	5	
%	7	E	6	e	5	
&	7	F	6 (5)	f	4	
'	4 (3)	G	7	g	5	
(	3	H	6	h	5 (6)	
)	3	I	3	i	3	
*	5	J	5	j	3 (4)	
+	5	K	7 (6)	k	5	
,	3	L	6	l	3	
—	4	M	8 (7)	m	7	
.	3	N	6	n	5	
/	4	O	7 (6)	o	5	
0	5	P	6 (5)	p	5 (6)	
1	5	Q	7 (6)	q	5	
2	5	R	7 (6)	r	4 (5)	
3	5	S	5	s	4	
4	5	T	6	t	4 (5)	
5	5	U	6	u	5 (6)	
6	5	V	6	v	5	
7	5	W	8 (7)	w	7	
8	5	X	7 (6)	x	5 (6)	
9	5	Y	7 (6)	u	5	DW-II, DW-IIB, DWP-410 shown (DWP-210 in parentheses)
:	3	Z	6	z	5	
;	3	[	3	{	3	
<	5	\	4	|	3	
=	5	]	3	}	3	
>	5	^	5	~	5	
?	5	_	5			

The character widths are defined in terms of microspaces. (If you're unfamiliar with how dot positioning is done, this might be a good time to read "Positioning Print on the Paper," which describes microspace units.) From the table, you can see that character widths range from three, for such characters as a period or comma, to eight, for such characters as a percent sign and uppercase W. Character widths are always defined in 1/60-inch units, even though your printer may be capable of spacing in 1/120-inch units.

LINE JUSTIFICATION

There's no problem at all in using proportional-spaced characters with a ragged-right margin, as shown in Figure 8-9. Just print a line of text using an average character width of about five microspaces per character. (Slightly less for the DMP-210.) Occasionally, lines will be longer than you desire, if there's an overabundance of wide characters, or smaller than normal if there are more narrow-width characters. In general, though, you can simply include the text you want printed in common BASIC LPRINT lines.

However, suppose that you want to justify the text shown in Figure 8-10 below. We'll describe an algorithm that does the task, using the text in the figure as an example:

Menus are used not only in posh restaurants, but in posh computer software. You've seen menus on Radio Shack software, but let's illustrate the use of them to jog your memory. Suppose that we have written an applications program to process weather data. When the program is first loaded, it may display a menu of functions that may be selected, as shown in figure 4-6. If entry 4 is desired, then the user types a '4', and a new menu of items related to 'ANNUAL WEATHER DATA' is displayed for further selection. This type of implementation is termed 'MENU-DRIVEN.' Menus provide an easy-to-use format that is very descriptive. This section should definitely be interpreted as a plus for menu use. (I have a brother-in-law in the menu-printing business.) Menu printing is easy, of

Figure 8-9. Daisy Wheel Proportionally-Spaced Type with Unjustified Edge

'The microcomputer boom is over,' analysts at Computer Market Prognostication and Rumor Co., Inc. announced today. 'There are currently over 300 manufacturers of microcomputers all claiming to be shipping units at the rate of 10,000 units per week. There are 40 million households in the U.S. and we've been checking every other one. We found micros in only 3 out of 1757 households. Clearly someone is distorting the facts.'

'The microcomputer boom is over,' analysts at Computer Market Prognostication and Rumor Co., Inc. announced today. 'There are currently over 300 manufacturers of microcomputers all claiming to be shipping units at the rate of 10,000 units per week. There are 40 million households in the U.S. and we've been checking every other one. We found micros in only 3 out of 1757 households. Clearly someone is distorting the facts.'

Figure 8-10. Text for Justification 2

1. Find the number of microspaces in a line based upon the position of the left and right margin. In this case it's 540 microspaces. (We're using a Daisy Wheel IIB or DWP-210 printer here, with 120 microspaces per inch. The DW-II and DWP-410 will have 60 microspaces per inch.)
2. From text to be printed, start counting character widths by word, including spaces.
3. When the total number of microspaces in the current line of words exceeds the number of microspaces in a line (in the word "Market") go back to the beginning of the previous word.
4. Find the number of microspaces in all words through the last word.
5. Subtract the number of microspaces in all words from the total number of microspaces per line.
6. The result of the subtraction is the number of dot columns left over.

This number of microspaces must be spread out between words on a somewhat orderly basis to avoid large gaps between any two words.

7. The result is a line of justified text. The text "Market Prognostication . . . " is left over and applies to the next line's justification procedure.

You can see that it's not an easy task to justify text with proportional spacing. To simplify things for your own programming, use the subroutine shown in Listing 8-5. It will take any string of up to 255 characters (assumed to be words), and justify it, based on the total number of microspaces per line and character widths.

```
100 CLEAR 1000
110 DIM CH(128)
120 DATA 6,3,4,6,5,7,7,3,3,3,5,5,3,5,3,4,5,5,5,5,5,5,5,5,5,5
130 DATA 3,3,5,5,5,5,7,7,6,7,6,6,6,7,6,3,5,7,6,7,6,7,6,7,7,5
140 DATA 6,6,6,7,7,7,6,3,4,3,5,5,5,5,5,5,5,5,4,5,5,3,3,5,3,7
150 DATA 5,5,5,5,4,4,4,5,5,7,6,5,5,3,3,3,5,0,0,0,0,0,0,0,4,5
160 DATA 0,5,6,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0
170 FOR I=0 TO 127
180 READ CH(I)
190 NEXT I
200 ZZ$="This is a test of proportional justication on Daisy Wheel printers
 including the DW-II, DWP-IIB, DWP-210, and DWP-410.  Change line 10070 to Z
U=1 for the DW-II. The subroutine justifies and prints from string ZZ$."
210 ZL=60: ZR=420: ZV=1
220 GOSUB 10000: IF ZZ$<>"" THEN 220 ELSE STOP
10000 ' DAISY WHEEL JUSTIFICATION SUBROUTINE FOR DW-II, DWP-210, DWP-410
10010 ' ENTRY: ZZ$=STRING
10020 '        CH()=CHARACTER WIDTH ARRAY
10030 '        ZR=RIGHT MARGIN IN 1/60THS INCH FROM EDGE
10040 '        ZL=LEFT MARGIN IN 1/60THS INCH FROM EDGE
10050 '        ZV=LINE SPACING 1=SINGLE, 2=DOUBLE, ETC.
10060 ' EXIT:  LINE PRINTED AND ZZ$=REMAINDER OF STRING
10070 ZU=2 'CHANGE TO ZU=1 FOR DW-II
10080 LPRINT CHR$(27);CHR$(17);
10090 ZB=ZR-ZL: ZC=0: ZS=0
10100 FOR ZI=1 TO LEN(ZZ$)
10110 ZD$=MID$(ZZ$,ZI,1): IF ZD$=" " THEN ZE=ZI: ZC=0: ZS=ZS+1: GOTO 10130
10120 ZC=ZC+CH(ASC(ZD$)-32)
10130 ZB=ZB-CH(ASC(ZD$)-32)
10140 IF ZB<=0 THEN 10180
10150 NEXT ZI
10160 IF ZL=1 THEN 10170 ELSE FOR ZI=1 TO ZL-1:LPRINT CHR$(27);CHR$(ZU);: N
EXT ZI
10170 LPRINT ZZ$: ZZ$="": GOTO 10310
10180 IF ZS=0 THEN 10310
10190 ZI=ZE: ZB=ZB+ZC+4+ZU: ZY$=RIGHT$(ZZ$,LEN(ZZ$)-ZI)
10200 ZZ$=LEFT$(ZZ$,ZI-1)
10210 ZS=ZS-1:ZF=ZB-INT(ZB/ZS)*ZS: ZB=INT(ZB/ZS)
10220 IF ZL=1 THEN 10240
10230 FOR ZI=1 TO ZL-1: LPRINT CHR$(27);CHR$(ZU);: NEXT ZI
10240 FOR ZI=1 TO LEN(ZZ$)
10250 ZD$=MID$(ZZ$,ZI,1): IF ZD$<>" " THEN LPRINT ZD$;: GOTO 10280
10260 FOR ZJ=1 TO ZB+4+ZU: LPRINT CHR$(27);CHR$(ZU);: NEXT ZJ
10270 IF ZF<>0 THEN LPRINT CHR$(27);CHR$(ZU);:ZF=ZF-1
10280 NEXT ZI
10290 LPRINT STRING$(ZV,13);
10300 ZZ$=ZY$
10310 RETURN
```

Listing 8-5. Daisy Wheel Proportional Justification Program

To use the subroutine, first change the DATA statements for the character widths of your own printer. The ones we've included here are for the DWP-210 daisy wheel, so if you have that printer you can leave the DATA lines as they are. Otherwise, enter a total of 160 values, one for each character from 32 through 127, zeroes for the "unused" characters from 128 through 159, and one each for values from 160 through 191 for the European character set (see "Using European Symbols"). If your printer does not have code for some of the European characters or other characters, enter a zero for the width. The widths must be in strict order as shown, otherwise some characters will be the wrong width.

To use the subroutine, put the text string you want justified in ZZ$, the left margin in microspaces in ZL, the right margin in microspaces in ZR, and the line spacing in ZV (ZV = 1 doesn't skip lines, ZV = 2 double spaces, and so forth). Then call the subroutine and the line will be printed, justified on the right with word wrap. The remainder of the text will be returned in ZZ$. Suppose that you had ten long strings of text to print, with a left margin of 120 microspaces, a right margin of 840 microspaces, and single line spacing. The strings are in a string array of AA$(1) through AA$(10). This code would print the text:

```
100 ZZ = "": ZL = 120: ZR = 840: ZV = 1
110 FOR I = 1 TO 10
120 ZZ$ = ZZ$ + AA$(1): GOSUB 10000
130 NEXT I
```

BOLDFACE PRINTING IN PROPORTIONAL SPACING

We discussed boldface printing in "Using Bold Printing" in Chapter 7 and said that it was difficult to backspace and overprint in bold when you have variable character widths. Knowing the widths, though, makes it a little easier. The subroutine shown in Listing 8-6 allows you to do your own bold printing of a string of text. You must enter the DATA values in the CH array, as before, for your own printer. Call the subroutine with the text to be printed in bold in ZZ$. The subroutine will print the text and return.

```
10000 'SUB BOLD PRINT FOR DAISY WHEEL PRINTERS
10010 'ZZ$ = STRING TO TYPE BOLD
10020 ZU=2 'CHANGE TO ZU=1 FOR DW-II
10030 LPRINT ZZ$;
10040 FOR ZJ=1 TO LEN(ZZ$)
10050 ZB=CH(ASC(ZZ$)-32)
10060 FOR ZI=1 TO ZB : LPRINT CHR$(8);CHR$(ZU); : NEXT ZI
10070 NEXT ZJ
10080 LPRINT ZZ$; : RETURN
```

Listing 8-6. Daisy Wheel Bold Space Proportional Mode Program

GENERAL BACKSPACE SUBROUTINE FOR PROPORTIONAL PRINTING

Sometimes you want to backspace proportional characters for overprinting. The subroutine in Listing 8-7 will allow you to backspace so you can slash zeroes or strike through other characters. Again, the CH array must contain character widths.

Enter the subroutine with the character for the backspace in ZZ$ and the character to be overprinted in ZY$. At the return from the subroutine, the print head will be positioned as if the first character alone had been printed. (The second character is assumed to be equal to or smaller than the first.) The width values for the characters must be in the CH array as before.

```
10000 'SUB STRIKE-OVER FOR DAISY WHEEL PRINTERS
10010 'ZZ$ = CHAR TO BE STRUCK OVER
10020 'ZY$ = CHAR TO STRIKE OVER
10030 ZU=2 'CHANGE TO ZU=1 FOR DW-II
10040 ZB=CH(ASC(ZZ$)-32) : FOR ZI=1 TO ZB : LPRINT CHR$(8);CHR$(ZU); : NEXT
 ZI : LPRINT ZY$;
10050 ZD=ZB-CH(ASC(ZY$)-32) : IF ZD=0 THEN RETURN
10060 IF ZD>0 THEN FOR ZI=1 TO ZD : LPRINT CHR$(27);CHR$(ZU); : NEXT ZI : R
ETURN
10070 ZD=-ZD : FOR ZI=1 TO ZD : LPRINT CHR$(8);CHR$(ZU); : NEXT ZI : RETURN
```

Listing 8-7. Daisy Wheel Proportional Backspace Program

EXTERNAL MODE FOR DAISY WHEELS

When you run a standard daisy print wheel in PS mode, the printer uses an internal width table to determine the character widths to use for spacing the characters. The standard widths are shown in Table 8-1. The printer reads the 10/12/PS switch on the front panel to determine whether PS mode is in effect. PS mode can also be set under software control by sending a CHR$(27);CHR$(17) escape sequence to the printer.

The standard spacing from the internal table in the printer's ROM is used regardless of what type of wheel you actually have mounted on the printer. You can actually run in proportional-spacing mode with a Courier wheel designed for 12 pitch, even though the resulting print will not look as nice as a PS typeface designed specifically for PS mode.

Certain daisy wheels don't fit the spacing requirements for either 10 or 12 pitch or for the standard proportional spacing widths. The "Cubic 15" type face, for example, is meant to be spaced at 15 characters per inch. If you were to use 12 pitch for this daisy-wheel, the type would have wide spaces between letters, as shown in Figure 8-11—not at all what the type designers had in mind.

This is a sample of CUBIC 15 at 12 pitch.

This is a sample of CUBIC 15 at 15 pitch.

Figure 8-11. Cubic 15 Spaced at 12 Pitch Vs. 15 Pitch

How do you adjust the spacing for wheels such as Cubic 15 and others? There's a special daisy-wheel printer mode called *external programming mode* which lets you define the widths for each character. Using the external programming mode, you can define widths from 0 through 15/60 of an inch for all characters on a print wheel. You can also define two additional parameters, *impression level* and *ribbon feed*.

The external programming mode must be run when the 10/12/PS front-panel switch is set to PS. Sending a CHR$(27);CHR$(24) starts this mode; sending a CHR$(27);CHR$(25) resets to the normal PS mode.

After the external programming mode is initiated, characters are typed by sending two codes for each single character to be typed, as shown in Figure 8-12. The first code is the normal code for the character. The second code is a special value from 0 through 255 that defines the width, ribbon feed, and impression level.

The character width and impression level are determined from a

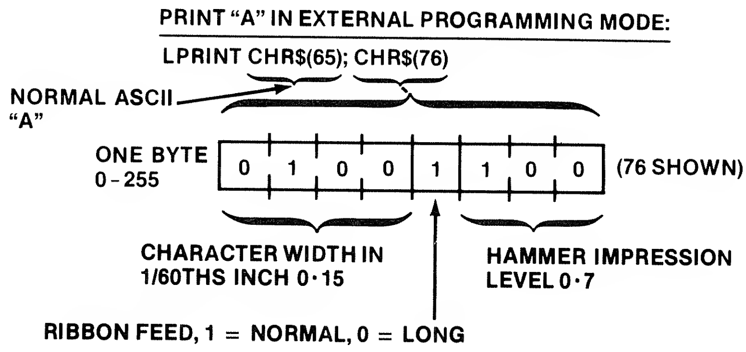


Figure 8-12. External Mode Printing

character table included with each daisy print wheel that requires external programming mode. A sample table is shown in Figure 8-13.

The width in the sample table is the width to use for each character for proper spacing. In this case, for Cubic 15, the widths are all 4/60, or 1/15 of an inch—what you would expect for a 15 pitch font.

The impression level, or *hammer set* determines how hard the print hammer will strike the daisy-wheel character. The smaller the character, the less force required to produce a print image. It's like hitting a wooden table top with a ball-peen hammer compared to a wooden mallet—the ball-peen hammer will leave a deeper impression in the wood if both are swung with the same force. The ribbon feed determines the amount the ribbon is advanced.

You can use the two codes given with the daisy wheel table for impression level and ribbon feed directly. In the case of the Cubic 15, for example, you'd print a "T" character in external program mode this way:

```
100 LPRINT CHR$(84);CHR$(76)
```

The 84 is the normal ASCII code for the T, and the 76 is the coded value for a character width of 4/60 of an inch, ribbon feed of 1, and impression level of 4.

Two codes must be sent for *every* character printed in external program mode.

You can also code the second character to be sent in external programming mode with your own widths, ribbon feed, and impression level. One caution: Go easy on the impression level. Higher impression levels mean the hammer strikes harder, which might result in more wear on the wheels. To code your own parameters, do this:

1. Determine a width value in 1/60-inch units from 0 through 15. Multiply it by 16.
2. Use a ribbon value of one, amounting to a value of eight in the coded character.
3. Use a reasonable impression level based on the values found for the same character in the proportional spacing charts (see Table 8-2). This should be a value of 0 through 7.
4. Add the values together to get the CHR$() value.

As an example, suppose you wanted to make all Ws extra wide, but not

Radio Shack

To use this font, you must use external program mode (see page 16 of 26-1158 Owner's Manual). Input corresponding codes as given below.

Pour utiliser cette fonte, vous devez être en mode de programme externe (voir page 16 du manuel de l'usager 26-1158). Entrez les codes correspondante tels que donnés ci-dessous.

87156121

26-1487 **26-1487**

87156121

NOTE : Impression control should be in the Low position.

NOTE : Réglage de la Force de Frappe "Low" (Faible).

	HEX	DEC		HEX	DEC		HEX	DEC		HEX	DEC
			@	40+4E	64+ 78	`	60+48	96+ 72	à	80+4D	128+ 77
!	21+4A	33+ 74	A	41+4C	65+ 76	a	61+4D	97+ 77	ç	9C+4C	156+ 76
"	22+4A	34+ 74	B	42+4E	66+ 78	b	62+4D	98+ 77	£	A3+4D	163+ 77
#	23+4E	35+ 78	C	43+4C	67+ 76	c	63+4C	99+ 76	µ	A5+4D	165+ 77
$	24+4E	36+ 78	D	44+40	68+ 77	d	64+40	100+ 77	°	A6+4A	166+ 74
%	25+4D	37+ 77	E	45+4D	69+ 77	e	65+4C	101+ 76	´	A7+48	167+ 72
&	26+4E	38+ 78	F	46+4C	70+ 76	f	66+48	102+ 75	†	A8+4C	168+ 76
'	27+49	39+ 73	G	47+4D	71+ 77	g	67+40	103+ 77	¯	A9+4C	169+ 76
(	28+48	40+ 75	H	48+4C	72+ 76	h	68+4C	104+ 76	●	AA+4D	170+ 77
)	29+48	41+ 75	I	49+48	73+ 75	i	69+48	105+ 75	○	A8+4D	171+ 77
*	2A+48	42+ 75	J	4A+48	74+ 75	j	6A+48	106+ 75	¼	AC+4D	172+ 77
+	28+4A	43+ 74	K	48+4E	75+ 78	k	68+4C	107+ 76	¾	AD+4D	173+ 77
,	2C+48	44+ 72	L	4C+48	76+ 75	l	6C+48	108+ 75	½	AE+4D	174+ 77
-	2D+48	45+ 72	M	4D+4E	77+ 78	m	6D+4E	109+ 78	¶	AF+4E	175+ 78
.	2E+48	46+ 72	N	4E+4C	78+ 76	n	6E+4C	110+ 76	é	88+4D	187+ 77
/	2F+48	47+ 75	O	4F+4D	79+ 77	o	6F+4C	111+ 76	ù	8C+4C	188+ 76
0	30+4C	48+ 76	P	50+40	80+ 77	p	7D+4D	112+ 77	è	8D+40	189+ 77
1	31+48	49+ 75	Q	51+4E	81+ 78	q	71+4D	113+ 77	¨	8E+48	190+ 72
2	32+4C	50+ 76	R	52+4D	82+ 77	r	72+48	114+ 75	ƒ	8F+4C	191+ 76
3	33+4C	51+ 76	S	53+4C	83+ 76	s	73+48	115+ 75	§	C0+40	192+ 77
4	34+4C	52+ 76	T	54+4C	84+ 76	t	74+48	116+ 75	¥	CC+40	204+ 77
5	35+4C	53+ 76	U	55+4C	85+ 76	u	75+4C	117+ 76	Ä	D8+40	219+ 77
6	36+4D	54+ 77	V	56+4C	86+ 76	v	76+48	118+ 75	Ö	0C+4E	220+ 78
7	37+4C	55+ 76	W	57+4E	87+ 78	w	77+4D	119+ 77	Ü	DD+4D	221+ 77
8	38+4D	56+ 77	X	58+4D	88+ 77	x	78+4C	120+ 76	¢	DF+4C	222+ 76
9	39+4D	57+ 77	Y	59+4C	89+ 76	y	79+4C	121+ 76	_	DF+44	223+ 68
:	3A+49	58+ 73	Z	5A+4C	90+ 76	z	7A+4C	122+ 76	ä	F8+4D	251+ 77
;	38+4A	59+ 74	[	58+48	91+ 75	{	78+48	123+ 75	ö	FC+4D	252+ 77
<	3C+48	60+ 75	\	5C+48	92+ 75	|	7C+48	124+ 75	ü	FD+4D	253+ 77
=	3D+48	61+ 75	]	50+48	93+ 75	}	7D+48	125+ 75	ß	FE+4E	254+ 78
>	3E+48	62+ 75	^	5E+49	94+ 73	~	7E+49	126+ 73			
?	3F+48	63+ 75	_	5F+40	95+ 64						

Printed in japan

Figure 8-13. External Mode Character Table

very dark. The maximum width would be 15/60 of an inch. Multiply this by 16 to get 240. Add a ribbon value of eight to get 248. Adding an impression level of 1 results in 249 as the second CHR$() value. The extra wide W would be printed by

```
100 LPRINT CHR$(87);CHR$(249)
```

Because the external programming mode is a lot of work, you may not want to use it for your own programs. However, it does give you complete control over the way characters are formed. The program in Listing 8-8 shows a proportional spacing justification program that works with an external programming mode typeface, in this case Cubic 15. Use it as an example for your own specifications. The CH$() array contains 96 character widths as before, but this time the impression level and ribbon feed are also included in the array values. The properly encoded values can be found on the character table included with each daisy print wheel. The one we're using here is shown in Figure 8-13.

Printer Hint

NOTES ON THE IMPRESSION LEVEL AND RIBBON FEED

Don't expect the impression level to have too big an impact (sorry...) on the darkness of the print. There is a detectable difference, but not as much as you might expect. It's probably best to stay at the values defined by your printer manual, to save wear and tear on the daisy-wheel.

The normal setting for ribbon feed (a one bit, or 16 value) sets a normal ribbon feed. A zero bit (16 not added to the external mode value) defines a long amount of ribbon. If you've noticed your daisy-wheel printer ribbon after printing, you may have observed that it over-prints to extend the ribbon life in the normal mode. This is fine for almost every application, but for producing camera-ready copy, you may want to go to a long feed of ribbon to get the cleanest copy possible.

Table 8-2. Impression levels DW-II, DW-IIB, DWP-410

(Space)		@	6	`	0
!	2	A	4	a	5
"	2	B	6	b	5
#	6	C	4	c	4
$	6	D	5	d	5
%	5	E	5	e	4
&	6	F	4	f	3
'	1	G	5	g	5
(	3	H	4	h	4
)	3	I	3	i	3
*	3	J	3	j	3
+	2	K	6	k	4
,	0	L	3	l	3
—	0	M	6	m	6
.	0	N	4	n	4
/	3	O	5	o	4
0	4	P	5	p	5
1	3	Q	6	q	5
2	4	R	5	r	3
3	4	S	4	s	3
4	4	T	4	t	3
5	4	U	4	u	4
6	5	V	4	v	3
7	4	W	6	w	5
8	5	X	5	x	4
9	5	Y	4	y	4
:	1	Z	4	z	4
;	2	[	3	{	3
<	3	\	3			3
=	3	]	3	}	3
>	3	^	1	~	1
?	3	_	0		

```
100 CLEAR 1000
110 DIM CH(128)
120 DATA 96,74,74,78,78,77,78,73,75,75,75,74,72,72,72,75,76,75,76,76,76,76,
77,76,77,77,73,74,75,75,75,75
130 DATA 78,76,78,76,77,77,76,77,76,75,75,78,75,78,76,77,77,78,77,76,76,76,
76,78,77,76,76,75,75,75,73,64
140 DATA 72,77,77,76,77,76,75,77,76,75,75,76,75,78,76,76,77,77,75,75,75,76,
75,77,76,76,76,75,75,75,73,0
170 FOR I=0 TO 95
180 READ CH(I)
190 NEXT I
200 ZZ$="This is a test of proportional justication on Daisy Wheel printers
 including the DW-II, DW-IIB, DWP-210, and DWP-410. Change line 10070 to ZU
=1 for the DW-II. The subroutine justifies and prints from string ZZ$."
210 ZL=60: ZR=220: ZV=1
220 GOSUB 10000: IF ZZ$<>"" THEN 220 ELSE STOP
10000 'EXTERNAL PROG. JUSTIFICATION SUB. FOR DW-II, IIB, DWP-210, 410
10010 ' ENTRY: ZZ$=STRING
10020 '        CH()=CHARACTER WIDTH ARRAY
10030 '        ZR=RIGHT MARGIN IN 1/60THS INCH FROM EDGE
10040 '        ZL=LEFT MARGIN IN 1/60THS INCH FROM EDGE
10050 '        ZV=LINE SPACING 1=SINGLE, 2=DOUBLE, ETC.
10060 ' EXIT:  LINE PRINTED AND ZZ$=REMAINDER OF STRING
```

```
10070 ZU=2 'CHANGE TO ZU=1 FOR DW-II
10080 LPRINT CHR$(27);CHR$(17);CHR$(27);CHR$(24);
10090 ZB=ZR-ZL: ZC=0: ZS=0
10100 FOR ZI=1 TO LEN(ZZ$)
10110 ZD$=MID$(ZZ$,ZI,1): IF ZD$=" " THEN ZE=ZI: ZC=0: ZS=ZS+1: GOTO 10130
10120 ZC=ZC+(CH(ASC(ZD$)-32) AND 240)/16
10130 ZB=ZB-(CH(ASC(ZD$)-32) AND 240)/16
10140 IF ZB<=0 THEN 10200
10150 NEXT ZI
10160 IF ZL=1 THEN 10170 ELSE FOR ZI=1 TO ZL-1:LPRINT CHR$(27);CHR$(ZU);: N
EXT ZI
10170 FOR ZI=1 TO LEN(ZZ$) : ZD$=MID$(ZZ$,ZI,1)
10180 LPRINT ZD$; : IF ZD$>" " THEN LPRINT CHR$(CH(ASC(ZD$)-32));
10190 NEXT ZI : ZY$="" : GOTO 10310
10200 IF ZS=0 THEN 10330
10210 ZI=ZE: ZB=ZB+ZC+4+ZU: ZY$=RIGHT$(ZZ$,LEN(ZZ$)-ZI)
10220 ZZ$=LEFT$(ZZ$,ZI-1)
10230 ZS=ZS-1:ZF=ZB-INT(ZB/ZS)*ZS: ZB=INT(ZB/ZS)
10240 IF ZL=1 THEN 10260
10250 FOR ZI=1 TO ZL-1: LPRINT CHR$(27);CHR$(ZU);: NEXT ZI
10260 FOR ZI=1 TO LEN(ZZ$)
10270 ZD$=MID$(ZZ$,ZI,1): IF ZD$<>" " THEN LPRINT ZD$;CHR$(CH(ASC(ZD$)-32))
; : GOTO 10300
10280 FOR ZJ=1 TO ZB+4+ZU: LPRINT CHR$(27);CHR$(ZU);: NEXT ZJ
10290 IF ZF<>0 THEN LPRINT CHR$(27);CHR$(ZU);:ZF=ZF-1
10300 NEXT ZI
10310 LPRINT CHR$(27);CHR$(25);STRING$(ZV,13);
10320 ZZ$=ZY$
10330 RETURN
```

Listing 8-8. External Programming Mode Proportional Program

CHAPTER 9

WORD PROCESSING APPLICATIONS

In this chapter, we'll apply some of the tricks we've learned to specific printing situations. We'll discuss in detail the printing of mailing labels and form letters. Later we'll cover printing text screens.

PRINTING MAILING LABELS

This is one of the easiest applications to do with a printer. Mailing labels come in a variety of sizes; Radio Shack currently carries one-wide, 4¼-inch; two-wide, 9½-inch; and three-wide, 9½-inch. Typical labels appear as shown in Figure 9-1.

The labels are spaced on the forms so that there's a vertical separation of one inch between labels. This represents six lines at a standard spacing of six lines per inch.

In the programs below we'll use standard line spacings of six lines per inch and a pitch of ten characters per inch. These are the normal settings that your printer should use after you turn it on—you won't have to perform any special actions for these conditions. In some cases, you may have to set some DIP or front-panel switches to get this spacing.

PRINTING ONE-WIDE LABELS

Insert the labels in your tractor feed or friction feed printer so that the first print position is over the "1" of the rule above the carriage, if your printer has a rule. If your printer does not have a rule, insert the labels near the left margin.

Use the following program in BASIC to print the same label over and over:

```
100 LPRINT "William Barden, Jr."
110 LPRINT "200 N. S. Memory Lane
120 LPRINT "Computer City, CA 92692"
130 LPRINT: LPRINT: LPRINT
140 GOTO 100
```

This program will print the three address lines, skip three lines, and then repeat the process. If your printer is set up for six-lines-per-inch line spacing, the printing will be spaced at one inch intervals. You may have to temporarily put the printer off line, and then paper feed to align the vertical

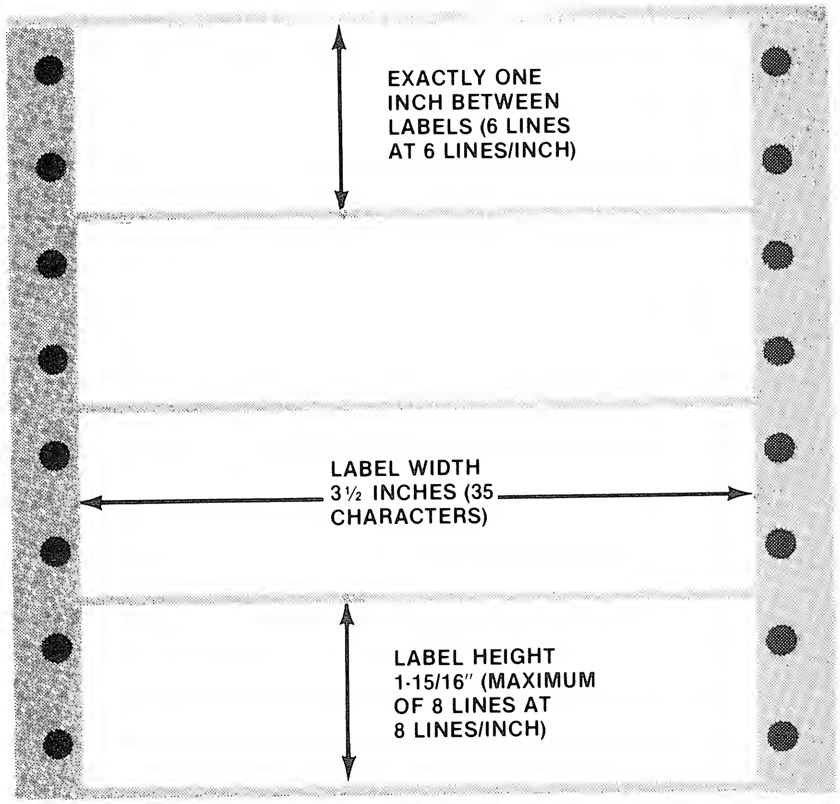


Figure 9-1. Mailing Labels

spacing of the labels. You may also have to add some leading spaces before printing, if your labels are in the middle of the carriage. You can do that one of two ways: An easy way is simply to use spaces before the text:

```
100 LPRINT "              William Barden, Jr."
110 LPRINT "            200 N.S. Memory Lane
120 LPRINT "          Computer City, CA 92692"
130 LPRINT: LPRINT: LPRINT
140 GOTO 100
```

Here we've not only added spaces to make the printing appear further along the line, but we've centered the text.

A second approach to adding spaces is to use the TAB command in BASIC. The TAB command will tab to a specified column position. If your labels start at about position 30 along your carriage (look at the rule above the carriage if your printer has one, or measure in inches and multiply by 10), you'd use TAB(30) before the text to be printed:

```
100 LPRINT TAB(30);"William Barden, Jr."
110 LPRINT TAB(30);"200 N. S. Memory Lane"
120 LPRINT TAB(30);"Computer City, CA 92692"
130 LPRINT: LPRINT: LPRINT
140 GOTO 100
```

If you want to print a fourth, or fourth and fifth line, just delete one or two of the LPRINTs and use text instead:

```
100 LPRINT TAB(30);"William Barden, Jr."
```

```
110 LPRINT TAB(30);"200 N. S. Memory Lane"
120 LPRINT TAB(30);"Computer City, CA 92692"
130 LPRINT TAB(30);"(714) 555-1212"
140 LPRINT: LPRINT
150 GOTO 100
```

PRINTING TWO-WIDE AND THREE-WIDE LABELS

The easiest way to do this is to spread out identical text along the same line, using tabs. For two-wide labels:

```
100 LPRINT "William Barden, Jr.";TAB(XX);"William Barden,Jr."
110 LPRINT "200 N. S. Memory Lane";TAB(XX);"200 N.S.
           Memory Lane"
120 LPRINT "Computer City, CA 92692";TAB(XX);
           "Computer City, CA 92692"
130 LPRINT: LPRINT: LPRINT
140 GOTO 100
```

Use a tab position of about 5 for XX and about 47 for YY. Three-wide labels use an additional TAB and text.

You can also substitute one or two different addresses, in the second or third position, to "gang print" two or three addresses at the same time. If the placement of the printing is off slightly, adjust the TAB values. If you'd like to center the text, you can do that also by adjusting the TAB values. Just remember that the total number of characters across is 85, in 8.5 inches, for normal 10-pitch spacing.

Press <BREAK> at any time to stop the BASIC program. Entering RUN will start the program again. Save the program on disk or cassette once you have the proper spacing.

USING DIFFERENT PITCHES OR LINE SPACING

You can use the same programs with different character pitches or line spacing. If you want to use 12 characters per inch, for example, TAB on that basis. If you used eight-lines-per-inch line spacing, you have room for seven LPRINT lines and could do this:

```
100 LPRINT "George Boole"
110 LPRINT "Logical Computer Data Digital Comm Company"
120 LPRINT "Mail Station 1254"
130 LPRINT "Highland Center Road"
140 LPRINT "State Route 54"
150 LPRINT "Annapolis, Maryland"
160 LPRINT "55555"
170 LPRINT
180 GOTO 100
```

Before printing, set the printer to the proper pitch and line spacing. See "Positioning Print on the Paper" to find out how to do this.

PRINTING BOILERPLATE AND FORM LETTERS

I hope the Bar Association or Postal Service doesn't get after us for this, but in this section we'd like to tell you how to easily print standard contracts, "boilerplate," and form letters. What I'm discussing here is any documentation that is the same except for a few areas in which the name and address, dollar amounts, or other short text or data change. I'm sure you've seen form letters in the mail. (Ours usually start "Dear Mr. Jr. Barden," although once I had a "Dear William $f654&&"—undoubtedly from another android . . .)

It's fairly easy to write a program to generate form letters, standard contracts, or other boilerplate. One that will work with a typical form letter is shown in Figure 9-2.

```
Computer Clearinghouse
10500 S. Swindle Lane
Silicon Valley South, California 92005

Mr. %%%&&%%% Barden, Jr.
200 N.S. Memory Lane
Computer City, CA 92692

Dear Mr. , Jr.:

You have won either a Mercedes-Benz 300SD, a color televi-
sion, or a ball point pen! All you have to do to collect your
prize, Mr. . Jr., is to call, toll free, (800) 555-1212 and
talk to one of our representatives. To qualify for the prize,
Mr. , Jr., simply buy over $99 worth of merchandise from our
catalog. If you'll take the time to look through the catalog,
now, Mr. , Jr., (or may I call you %%%&&%%%?) you`ll find SUPER
deals on just about every type of old computer you've ever
wanted. $149.95 for a complete Univac I! How about a Harvard
Whirlwind computer for $149.95!! Would you believe a used
Cray I, for only $999.95!!!

These are just a few bargins you'll find from us.

Call that number, now, %%%&&!!!. Or we'll stuff your mailbox
from Monday through Saturday with 3-pound catalogs!

                        Warmest personal regards,



                        Carmine O'Flaherty, Sales Manager
```

Figure 9-2. Typical Form Letter

The bulk of the form letter is canned text. There's a return address which never changes, and the body of the form itself. There's also a "To"address, which we've designated as five "&&F" text strings. The "&&F"stands for Field—a text area to be filled in by the user. Inside the body of the form are other fields, which have the same format—two "&" characters and the letter F.

The program operates this way: It will take each line of text to be printed and print it, exactly as it appears. However, each time the program encounters &&F characters, it will pause and then prompt you to enter the proper characters for the field. After you've entered the text for the field, the program will print out the text, and then print any remaining text up to the next &&F field.

In addition to printing plain text, the program uses these characters to indicate functions:

&&U	Start Underline
&&E	End Underline
&&B	Start Boldface
&&N	End Boldface
&&A	Above one half line (for superscripting)
&&D	Down one half line (for subscripting)
&&P	New page
&&L	Skip Line
&&S	Stop, page eject, and prepare for new letter

These functions can occur anywhere in the text. However, an underline, boldface, superscript, or subscript must start and end on the same line (start again on the next line if necessary). Do not use the functions unless your printer is equipped to handle them.

The condensed form of the form letter, using these function codes, is shown in Figure 9-3. To make up your own form letter, follow the sample here. All lines of the text must be put into the form ZZ$(1)= "xxx xxxx xxx" through ZZ$(NN)= "xxx xxx xxxxx", where NN is the last line of the text and the x's are text characters. The last line of text is ended by ZZ$(NN)= "&&S". Each ZZ$() string is a separate line to be printed. This program is meant to be used with dot-matrix printers that have word-processing mode or with daisy-wheel printers unless the superscripting and subscripting functions are not used, in which case it can be used with any printer.

Note: Change the number of characters per line in the printer driver (see "Setting Top-of-Form . . . ") to 255 if you are doing a great deal of bold printing on a daisy-wheel printer. The BASIC printer driver counts all characters, even backspaces, in figuring line length and may do an automatic new line after many backspaces.

The program can be used with proportionally spaced characters, but the lines may not be right-justified for lines containing fields. The program does not justify lines—you'll have to adjust the ZZ$() strings for the best appearance to produce a master form letter or contract.

The dialogue of the program appears as shown in Figure 9-4, for our sample form letter:

The FORM program is shown in Listing 9-1. The program starts at line 10000. Append it to your own BASIC lines which define the master text to be printed.

Printer Hint

HOW THE FORM PROGRAM WORKS

The FORM program works with the strings in ZZ$(1) through ZZ$(NN). Each entry in the ZZ$ array represents one line to be printed. The program scans each entry, one character at a time. If the current character, and the next, are not equal to "&&", then the program assumes that the character is not a start of a control code and prints the first character. If the two characters are equal to "&&", then a special action is taken to send out the proper escape sequence for the control code. These actions are all fairly straightforward except for bold printing on a daisy-wheel printer. In this case the backspace subroutine is called by the main program to do the backspacing required for the daisy wheel. The backspace subroutine is similar to the one shown in "Backspacing and Strikethroughs for Fixed Pitch."

```
90 CLEAR 2000
100 'FORM PRINT PROGRAM
110 DIM ZZ$(100)
120 CLS: LPRINT CHR$(20) 'SET WORD PROCESSING MODE FOR SUB/SUPERSCRIPT
130 ZZ$(1)="Computer Clearinghouse"
140 ZZ$(2)="10500 S. Swindle Lane"
150 ZZ$(3)="Silicon Valley South, California 92005"
160 ZZ$(4)="&&L"
170 ZZ$(5)="&&F"
180 ZZ$(6)="&&F"
190 ZZ$(7)="&&F"
200 ZZ$(8)="&&L"
210 ZZ$(9)="Dear &&F"
220 ZZ$(10)="&&L"
230 ZZ$(11)="You have won either a &&BMercedes-Benz 300SD&&N, a &&Bcolor televi&&N
-"
240 ZZ$(12)="&&Bsion&&N, or a ball point pen! All you have to do to &&Ucollect&&E
your"
250 ZZ$(13)="prize, &&F, is to call, toll free, &&F and"
260 ZZ$(14)="talk to one of our representatives. To qualify for the prize,"
270 ZZ$(15)="&&F, simply buy over $99 worth of merchandise from our"
280 ZZ$(16)="catalog. If you'll take the time to look through the catalog,"
290 ZZ$(17)="now, &&F, (or may I call you &&F?) you'll find &&BSUPER&&N"
300 ZZ$(18)="deals on just about every type of old computer you've ever"
310 ZZ$(19)="wanted. &&U$149.95&&E for a complete &&BUnivac I&&N! How about a &&BH
arvard&&N"
320 ZZ$(20)="&&BWhirlwind computer&&N for &&U$149.95&&E!! Would you believe a used
"
330 ZZ$(21)="&&BCray I&&N, for only &&U$999.95&&E!!!"
340 ZZ$(22)="&&L"
350 ZZ$(23)="These are just a &&Bfew bargains&&N you'll find from us."
360 ZZ$(24)="&&L"
370 ZZ$(25)="Call that number, now, &&F!!! &&BOr we'll stuff your mailbox&&N"
380 ZZ$(26)="&&Bfrom Monday through Saturday with 3-pound catalogs&&N!"
390 ZZ$(27)="&&L"
400 ZZ$(28)="                              Warmest personal regards,"
410 ZZ$(29)="&&L"
420 ZZ$(30)="&&L"
430 ZZ$(31)="                              Carmine O'Flaherty, Sales Manager"
440 ZZ$(32)="&&S" 'indicates end of data
10000 ' PRINT FORMS PROGRAM. PRINTS FORM LETTER FROM ZZ$ ARRAY
10010 ' WITH SPECIAL && CODES FOR UNDERLINE, BOLD, SUPER-
10020 ' OR SUBSCRIPTING, NEWK'I
```

Figure 9-3. Condensed Form of Form Letter

PRINTING TEXT SCREENS

There are applications programs to print the screen for most Radio Shack computer systems. However, it's also possible to print the screen from within a BASIC program by a short BASIC subroutine. We'll describe screen print programs in BASIC here that will print text only. You'll find a corresponding program in Section III for printing graphics screens.

MODEL I, III, AND 4 SCREEN PRINT

In these computers, the screen display area uses RAM memory locations 15360 through 16383, for a total of 1024 screen characters (Model 4 in Model III mode). It's possible to PEEK and POKE these locations like any

PROGRAM DISPLAYS LINES AS IT TYPES THEM

```
Computer Clearinghouse
10500 S. Swindle Lane
Silicon Valley South, California 92005
```

PROGRAM ASKS FOR USER TEXT

```
&&F?Mr. William Barden, Jr.
```

(ENTER AFTER TEXT)

```
&&F?200 N.S. Memory Lane

&&F?Computer Cit
```

Figure 9-4. Form Letter Dialogue

others. (PEEK and POKE are two BASIC commands that allow you to access an RAM memory locations to write data or to read data.)

What characters are stored in video RAM depends upon what is being displayed upon the screen and it may range from normal text (ASCII) to graphics characters. The program below PEEKs (reads) all video RAM locations, discards any character that is not printable (not a code of 32 through 127), and then prints 16 lines, representing the 16 lines of the screen:

```
10000 ' MODEL I,III,4 16 BY 64 TEXT SCREEN DUMP
10010 ' PRINTS SCREEN DOUBLE-SPACED ON PRINTER
10020 FOR ZR=0 TO 15
10030 FOR ZC=0 TO 63
10040 ZB=PEEK(15360+(ZR*64)+ZC)
10050 IF(ZB<32) OR (ZB>127) THEN LPRINT " ";: GOTO 10070
10060 LPRINT CHR$(ZB);
10070 NEXT ZC
10080 LPRINT: LPRINT
10090 NEXT ZR
10100 RETURN
```

To use this program, simply enter it, together with your own BASIC program or with another BASIC program. Insert a

```
GOSUB 10000
```

at every place in the BASIC program where you want the screen printed. The screen display will automatically be printed out when the subroutine at 10000 is executed by the GOSUB.

An actual printout is shown in Figure 9-5. Note that the program double spaces lines to get proportions closer to the actual screen ratio of 4 to 3.

Printer Hint

HOW THE MODEL I, III, 4 SCREEN PRINT PROGRAM WORKS

This program uses variables ZR (row) and ZC (column) in two loops. A PEEK command in the inner loop looks at the video memory location to see which character is on the screen. The video memory location is defined by the row and column values in ZR and ZC and the start of the video memory at 15360. If the video memory value is less than a space or greater than the last text character (127), a blank character is printed, otherwise, the actual character in the video memory location is printed. After each row of 64 characters is printed, two line feeds double-space the rows. The program returns to the calling program after the 16 rows have been printed.

MC-10 AND COLOR COMPUTER SCREEN PRINT PROGRAM

See "Using the TP-10 to Print the Screen . . . " in Section III, if you have a TP-10 printer.

The Color Computer and MC-10 also use a portion of RAM memory to store the text screen. The screen characters are similar, but not identical, to the ASCII characters printed on Radio Shack printers. Certain ranges of characters are displayed in *inverse video*—black on green instead of green

```
10000 ' PRINT FORMS PROGRAM. PRINTS FORM LETTER FROM ZZ$ ARRAY
10010 ' WITH SPECIAL && CODES FOR UNDERLINE, BOLD, SUPER-
10020 ' OR SUBSCRIPTING, NEW PAGE, SKIP LINE, AND STOP.
10030 ' APPEND TO THE END OF YOUR MASTER BASIC CODE.
10040 ZX=0: Z2$="": ZP=12 '***USE ZP=12 FOR DAISY WHEEL***
10050 IF ZX=>100 THEN 10350 'NOTE: CHANGE ZX LIMIT TO MATCH ZZ$ DIM STMT
10060 ZX=ZX+1
10070 IF Z2$<>"L" THEN PRINT "": LPRINT ""
10080 ZL=LEN(ZZ$(ZX))
10090 IF ZL=0 THEN 10050
10100 ZY=0 'NEW LINE
10110 IF ZY=>ZL THEN 10050
10120 ZY=ZY+1
10130 IF MID$(ZZ$(ZX),ZY,2)<>"&&" THEN 10320
10140 Z2$=MID$(ZZ$(ZX),ZY+2,1)
10150 IF Z2$="U" THEN PRINT CHR$(15);: LPRINT CHR$(15);: GOTO 10290
10160 IF Z2$="E" THEN PRINT CHR$(14);: LPRINT CHR$(14);: GOTO 10290
10170 IF Z2$="A" THEN LPRINT CHR$(27);CHR$(30);: GOTO 10290
10180 IF Z2$="D" THEN LPRINT CHR$(27);CHR$(28);: GOTO 10290
10190 IF Z2$="L" THEN PRINT CHR$(13);: LPRINT CHR$(13);: GOTO 10290
10200 IF Z2$="P" THEN PRINT CHR$(12): LPRINT CHR$(12): GOTO 10290
10210 IF Z2$<>"B" THEN 10240
10220 IF ZP>0 THEN GOSUB 10410: GOTO 10290
10230 PRINT CHR$(27);CHR$(31);: LPRINT CHR$(27);CHR$(31);: GOTO 10290
10240 IF Z2$<>"N" THEN 10260
10250 PRINT CHR$(27);CHR$(32);: LPRINT CHR$(27);CHR$(32);: GOTO 10290
10260 IF Z2$<>"F" THEN 10310
10270 PRINT "": LINE INPUT "&&F?";ZE$
10280 LPRINT ZE$;
10290 ZY=ZY+2
10300 GOTO 10110
10310 IF Z2$="S" THEN 10350
10320 PRINT MID$(ZZ$(ZX),ZY,1);
10330 LPRINT MID$(ZZ$(ZX),ZY,1);
10340 GOTO 10110
10350 PRINT CHR$(12): LPRINT CHR$(12)
10360 PRINT "PRESS R TO RESTART"
10370 Z$=INKEY$
10380 IF Z$="" THEN 10370
10390 IF NOT (Z$="R" OR Z$="r") THEN 10370
10400 RETURN
10410 'SUB BACKSPACE FOR DAISYWHEEL PRINTER
10420 ZY=ZY+3
10430 ZD=ZY
10440 IF MID$(ZZ$(ZX),ZD,2)="&&" THEN ZD=ZD-1: GOTO 10480
10450 IF ZD=>ZL THEN 10480
10460 ZD=ZD+1
10470 GOTO 10440
10480 ZD=(ZD-ZY)+1
10490 PRINT MID$(ZZ$(ZX),ZY,ZD);: LPRINT MID$(ZZ$(ZX),ZY,ZD);
10500 ZZ=ZD*ZP
10510 ZA=ZZ-INT(ZZ/9)*9: ZB=INT(ZZ/9)
10520 IF ZB=0 THEN 10540
10530 FOR ZI=1 TO ZB: LPRINT CHR$(8);CHR$(9);: NEXT ZI
10540 IF ZA<>0 THEN LPRINT CHR$(8);CHR$(ZA);
10550 LPRINT MID$(ZZ$(ZX),ZY,ZD);
10560 ZY=ZY+ZD
10570 RETURN
```

Listing 9-1. FORM program

```
>LIST
10000 ' MODEL I,III,4 16 BY 64 TEXT SCREEN DUMP
10010 ' PRINTS SCREEN DOUBLE-SPACED ON PRINTER
10020 FOR ZR=0 TO 15
10030 FOR ZC=0 TO 63
10040 ZB=PEEK(15360+(ZR*64)+ZC)
10050 IF (ZB<32) OR (ZB>127) THEN LPRINT " ";: GOTO 10070
10060 LPRINT CHR$(ZB);
10070 NEXT ZC
10080 LPRINT: LPRINT
10090 NEXT ZR
10100 RETURN
READY
>
>RUN
```

Figure 9-5. Model I, III, 4 Screen Print Example

Printer Hint

HOW THE COLOR COMPUTER SCREEN PRINT PROGRAM WORKS

This program uses variables ZR (row) and ZC (column) in two loops. A PEEK command in the inner loop looks at the video memory location to see which character is on the screen. The video memory location is defined by the row and column values in ZR and ZC and the start of the text screen at 1024. If the video memory value is greater than 127, it is a graphics character and a space is substituted. If the value is less than 32, it is changed to an ASCII value by adding 32. If the value is greater than 95, it is changed to the proper ASCII value by subtracting 64. The value is then printed by a CHR$() print. After each row of 32 characters is printed, a line feed is done. After printing 16 rows, the program returns to the calling program.

on black. In addition, you may have block graphics characters displayed on the screen—these must be discarded in any print.

The text screen in the Color Computer is located at RAM locations 1024 through 1535. The program below PEEKs each of these locations, discards any non-printable character, translates from inverse video to the proper ASCII character, and then prints 16 lines of 32 characters each.

```
10000 ' COLOR COMPUTER 16 BY 32 TEXT SCREEN DUMP
10010 FOR ZR=0 TO 16
10020 FOR ZC=0 TO 31
10030 ZA=PEEK(1024+(ZR*32)+ZC)
10040 IF ZA >127 THEN ZA=32
10050 IF ZA<32 THEN ZA=ZA+32
10060 IF ZA >95 THEN ZA=ZA-64
10070 PRINT#-2,CHR$(ZA);
10080 NEXT ZC
10090 PRINT#-2
10100 NEXT ZR
10110 RETURN
```

To use this program, simply enter it, together with your own BASIC program, or with another BASIC program. Insert a

```
GOSUB 10000
```

at every place in the BASIC program where you want the screen printed.

The screen display will be automatically printed out when the subroutine at 10000 is executed by the GOSUB.

An actual printout is shown in Figure 9-6. The 3.2- by 2.66-inch dimensions are close to the 4 to 3 aspect ratio of the screen.

```
10000 ' COLOR COMPUTER 16 BY 32
TEXT SCREEN DUMP
10010 FOR ZR=0 TO 15
10020 FOR ZC=0 TO 31
10030 ZA=PEEK(1024+(ZR*32)+ZC)
10040 IF ZA>127 THEN ZA=32
10050 IF ZA<32 THEN ZA=ZA+32
10060 IF ZA>95 THEN ZA=ZA-64
10070 PRINT#-2,CHR$(ZA);
10080 NEXT ZC
10090 PRINT#-2
10100 NEXT ZR
10110 RETURN
OK
RUN
```

Figure 9-6. Color Computer/MC-10 Screen Print Example

MC-10 SCREEN PRINT PROGRAM

Use the program above for the MC-10, however, change line 10030 to read:

```
10030 ZA = PEEK(16384 + (ZR*32) + ZC)
```

Printer Hint

CUSTOM TAILORING THE SCREEN PRINT PROGRAMS

You can easily modify the programs above and custom tailor the screen print to your own specifications. You might want to spread the printing out, with a blank between each character, or print in double-width mode, or perhaps you might want to vary the spacing between lines to produce a printout with a closer approximation to the screen. Use the appropriate control codes, either before the printing takes place, or in the subroutine at the beginning, with a "reset" at the end. To get a double-width mode printout, for example, you'd add these two lines:

```
10005 LPRINT
        CHR$(27);CHR$(14)
'Note: "New" Codes
10095 LPRINT
        CHR$(27);CHR$(15)
```

Experiment with different combinations of things, if you know a little BASIC. Even if you don't know BASIC very well, you won't hurt anything by changing the code slightly!

SECTION 3

Graphics

CHAPTER 10

GRAPHICS PRINTING

In this section we'll look at graphics printing. As in Section II on text printing, this material is organized from simple topics to more complex topics. It includes detailed information on how to use graphics modes, and also more general topics about graphics applications, such as printing graphics screens and designing your own character sets.

This section is primarily for users with dot-matrix printers, although owners of daisy-wheel printers may want to read the material on plotting, which describes how to draw lines and graphs.

Note: As in Section II, we'll use the LPRINT command in some cases to stand for both the LPRINT (Model I, II, etc.) and the PRINT#-2, command used in the Color Computer and MC-10. Just substitute PRINT#-2, (don't forget the comma) for the LPRINT if using the programs for the Color Computer or MC-10.

BLOCK GRAPHICS

The block graphics characters can be used to draw horizontal and vertical lines, forms, and simple figures. Block graphics are available on all newer printers, including the LPV, VI, VIII, DMP-120, -200, -400, -420, -500, and -2100. There are no block graphics on daisy-wheel printers.

GENERAL DESCRIPTION

The block graphics characters are not included in the standard ASCII characters, but are part of the upper 128 set of characters in the range of codes from 128 through 255. The actual codes used for block graphics characters are codes 224 through 254, and are shown in Table 10-1.

There are three groups of block graphics, as you can see from the table. The first group contains the actual block graphics characters themselves — they are all of the 16 shapes that can be made by combining elements of a two by two block. These graphics are not related to the graphics blocks you'll see on the screen of a Model I, III, or 4/III. They are similar to the lower-resolution graphics modes in the Color Computer and MC-10, though.

Table 10-1. Block Graphics Characters

224	(blank)
225	▘
226	▝
227	▖
228	▗
229	▚
230	▞
231	▀
232	▄
233	▌
234	▐
235	▛
236	▜
237	▙
238	▟
239	█
240	┌
241	─
242	┐
243	┬
244	├
245	│
246	└
247	┘
248	┴
249	┤
250	┼
251	◤
252	◢
253	◥
254	◣

The second group of block graphics contains line segments. These are short line segments that can be used to draw horizontal and vertical lines, and to connect horizontal and vertical lines.

The third group of block graphics contains four filled-in triangles.

The most important thing to remember about the block graphics is that you don't have to be in graphics mode to use them! In fact, if you are in graphics mode, you won't be able to use them. The block graphics characters are printed by simply LPRINTing a block graphics character, by using a CHR$(). A block graphics character will occupy one character position on the paper, the same as any other printable character.

You must also not be in a correspondence or proportional-spacing character set when using block graphics. The exception to this is the DMP-2100. You can use the block graphics characters in any pitch also. If you're in 12 pitch (compressed characters on most printers), you'll get 12 block graphics characters or line segments per inch. If you're in 16.7 pitch (condensed characters on most printers), you'll get 16.7 block graphics characters or line segments per inch.

DRAWING HORIZONTAL LINES USING BLOCK GRAPHICS

To draw horizontal lines, use the 241 code in the block graphics character set. This is a short horizontal line segment that occupies the middle of the character position. To draw a horizontal line across the width of the paper, for example, do this:

```
100 LPRINT CHR$(28);CHR$(80);CHR$(241)
```

This statement draws 80 horizontal line segments, or an 8-inch horizontal line at ten characters per inch, using the repeat code.

To draw a horizontal line at any position, TAB first and then draw the line:

```
100 LPRINT TAB(42);CHR$(28);CHR$(40);CHR$(241)
```

This statement draws a four-inch horizontal line from a point 4.2 inches from the left margin.

DRAWING VERTICAL LINES USING BLOCK GRAPHICS

The block graphics character code 245 is used to draw vertical lines. This character is a short vertical line segment in the middle of a character position. The length of the line segment is one-half of the character position height, though. For this reason, you can't use a full line spacing for drawing a vertical line segment—you must first set a half forward line feed by a CHR$(27);CHR$(28). If you use full line spacing, you'll get a dashed vertical line as shown in Figure 10-1. Setting the half forward line feed has to be done only once, at the beginning of a program to draw lines. (Reset back to full line spacing by CHR$(27);CHR$(54).)

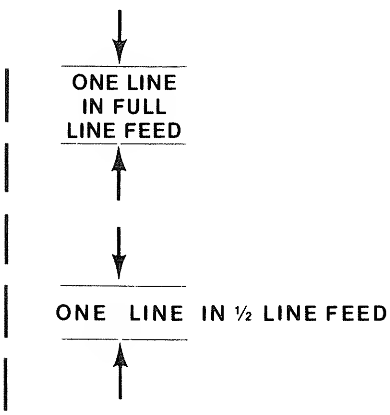


Figure 10-1. Full Line Spacing on Vertical Line Segment

This code draws a vertical line of 80 segments, or about 6⅔ inches:

```
100 LPRINT CHR$(27);CHR$(28)
110 FOR I=1 TO 80
120 LPRINT CHR$(245)
130 NEXT I
```

We couldn't use a repeat code above because it would have given us 80 vertical bars across one line, rather than 80 characters on 80 separate lines.

DRAWING BOXES AND FORMS

The vertical and horizontal line segments can be used together to draw a box or rectangle, and that's where the remainder of the line segments come in. The horizontal and vertical line segments are in the exact center of a character position. A *right-angle* character code of either 240, 242, 246, or 247 can connect a vertical and horizontal line at any corner, as shown in Figure 10-2. Furthermore, a vertical line can be joined to a midpoint on a horizontal line, or vice versa, by using the character codes of 243, 244, 248, and 249. Finally, a *cross* character (250) is the intersection point for two crossed lines.

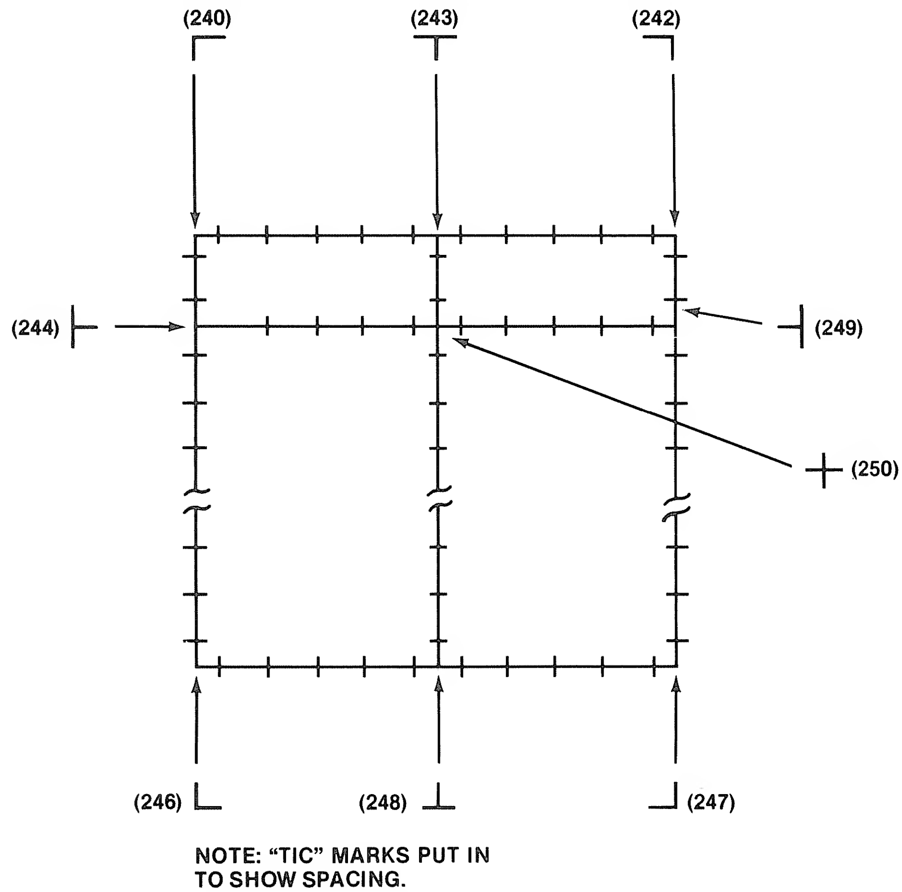


Figure 10-2. Block Graphic Line Segments

Using these character codes, it's fairly easy to draw a box or form. Suppose we wanted to draw the simple form shown in Figure 10-3. The first step in producing such a program is to lay it out on a graphics layout sheet that shows character positions. If you are using a 10 pitch (ten characters per inch) and 12 lines per inch spacing (don't forget the "half line" requirement), you'd want to use a layout sheet with identical spacing. Figure 10-4 shows the layout.

Printer Hint

BUSINESS FORMS RULER

One handy item to have when working with printers is a *business forms ruler.* It can be found in data-processing supply catalogs or at well-equipped stationary stores. I stole the one I use from my wife who, in another incarnation, was a business systems computer analyst, and it's great. It has 16 inches marked off in 1/10-inch segments (10-pitch), another scale in sixths and 12ths (full and half line spaces), another in 5/32nds (haven't found a use for that yet), and a normal inch-ruler scale.

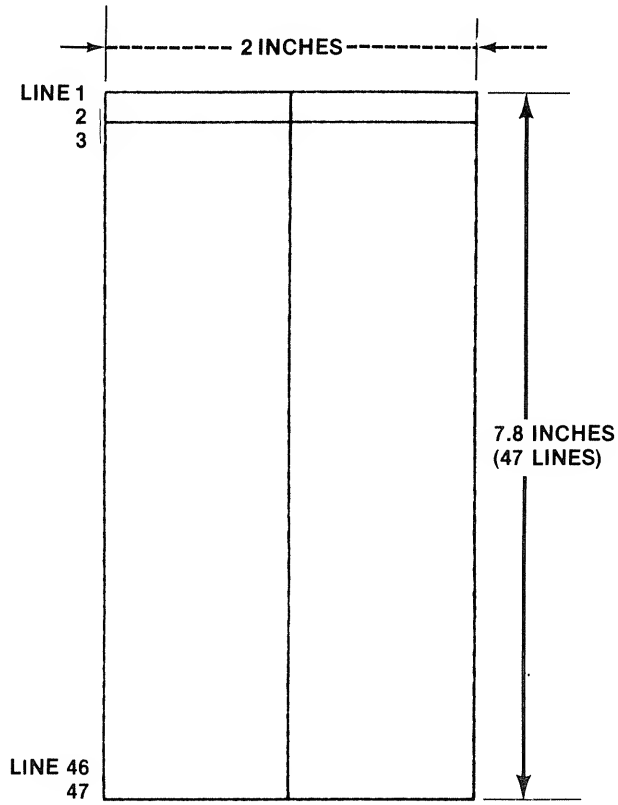


Figure 10-3. Form Example

The simplest way to print such a form is by the brute force method. From the layout figure, you can see that there will be 47 lines to be printed. The basic print program will look like this:

```
100 LPRINT CHR$();CHR$();CHR$() . . .
110 LPRINT CHR$();CHR$();CHR$() . . .
120 .
130 .
140 .
.
.
.
560 LPRINT CHR$();CHR$();CHR$() . . .
```

Each line will have 21 character positions, and you'll have to go over the layout sheet and find the corresponding block graphics character for the line segment and use that code in the CHR$ code. All told, there'll be 987 CHR$() codes! This is too much work for anyone with a computer, although it will produce the form.

A better approach is to do a little analysis of the form. Are there segments that can be repeated? The first thing you'll probably notice with this form is that lines 4 through 46 are the same. Why not do a simple loop here for 43 times to repeat that data? It would look like this:

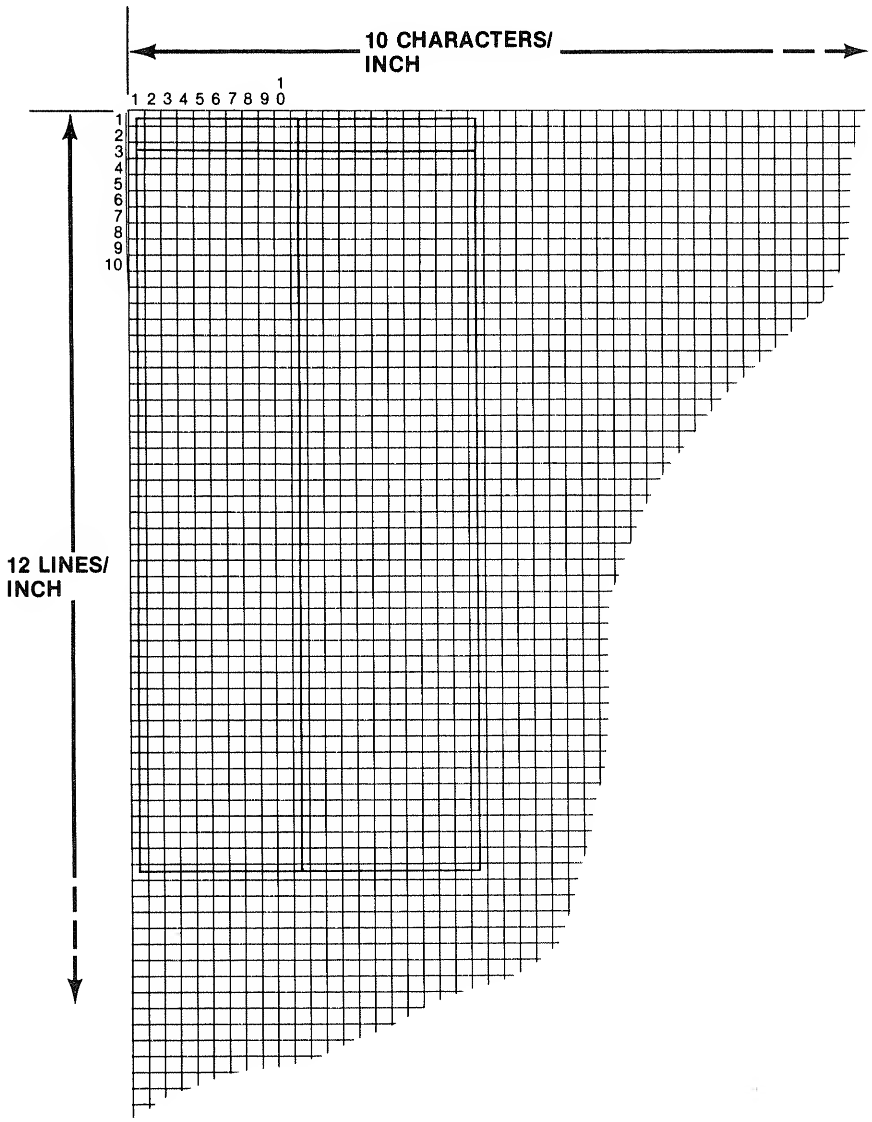


Figure 10-4. Form Layout

```
130 FOR I = 1 TO 43
140 LPRINT CHR$();CHR$();CHR$() . . .
150 NEXT I
```

We didn't fill in the CHR$ values yet, but we will. The entire program now looks like this:

```
100 LPRINT CHR$();CHR$();CHR$() . . .
110 LPRINT CHR$();CHR$();CHR$() . . .
120 LPRINT CHR$();CHR$();CHR$() . . .
130 FOR I = 1 TO 43
140 LPRINT CHR$();CHR$();CHR$() . . .
150 NEXT I
160 LPRINT CHR$();CHR$();CHR$() . . .
```

The first three lines print lines 1, 2, and 3, the next three lines print lines 4 through 46, and the last line prints line 47 of the form. Is there anything else we can simplify? Each of the 43 lines has only three printing characters, with a series of spaces in between. We could use either a TAB, a Repeat, a string constant, or a STRING$ function for this:

```
140 LPRINT CHR$(245);TAB(10);CHR$(245);TAB(20);CHR$(245)

140 LPRINT CHR$(245);CHR$(28);CHR$(9);CHR$(32);CHR$(245);
CHR$(28);CHR$(9);CHR$(32);CHR$(245)

140 LPRINT CHR$(245); "         ";CHR$(245);"         ";
CHR$(245)

140 LPRINT CHR$(245);STRING$(9," ");CHR$(245);STRING$(9" ");
CHR$(245)
```

We can use the same technique for the remaining four lines. The final result looks like this:

```
90 LPRINT CHR$(27);CHR$(28)
100 LPRINT CHR$(240);STRING$(9,CHR$(241));CHR$(243);
STRING$(9,CHR$(241));CHR$(242)
110 LPRINT CHR$(245);"         ";CHR$(245);"         ";
CHR$(245)
120 LPRINT CHR$(244);STRING$(9,CHR$(241));CHR$(250);
STRING$(9,CHR$(241));CHR$(249)
130 FOR I=1 TO 43
140 LPRINT CHR$(245);"         ";CHR$(245);"         ";
CHR$(245)
150 NEXT I
160 LPRINT CHR$(246);STRING$(9,CHR$(241));CHR$(248);
STRING$(9,241));CHR$(247)
```

This code is at least quite a bit shorter than 987 separate CHR$ values, although it is a little more complex. You can't go too far wrong in using these techniques, because it's easy to see the results and make slight adjustments to the code if required.

The same approach can be followed for any form — lay out the form first, and then look for lines that are similar and which can be shortened by using the repeat function, text strings for blanks, or STRING$ expressions. Another good idea is to make similar lines into subroutines. See your BASIC manual for this.

A FORMS PROGRAM FOR LAZY USERS

Okay, I know that you want to leave programming to programmers and don't want to muck around in it . . . The program below will save you a great deal of programming time and draw a form for you. All you have to do is enter the coordinates of line intersections. The program in Listing 10-1 will work with character positions up to 132 columns wide, and with up to 132 half-lines per page. It isn't a fast program, but it will do the job.

The program requires a system with at least 32K of user RAM memory, otherwise you'll get an "out of memory" error. You must first change TA in line 170 (Listing 10-1) to the value of a protected area in memory in which

the program data is to reside. Use a value of 14768 for a 32K Color Computer, 47536 for a 64K Color Computer, 31152 for a 48K Model I, III, 4, or 4P, or leave the program as is for a 64K Model I, III, 4, or 4P. In loading BASIC, protect this area *minus one* by setting Memory Size (I, III, 4, or 4P) or doing a CLEAR 100,NN in the Color Computer.

```
100 'FORM GENERATOR PROGRAM
110 'PROTECT MEMORY AT TA-1
120 CLS : PRINT"FORM PROGRAM" : PRINT
130 CLEAR 1000
140 DEFINT A-Z 'FOR MODEL I,III,4
150 DATA 245,241,246,245,245,240,244,241,247,241,248,242,249,243,250
160 FOR X=1 TO 15 : READ S : S$=S$+CHR$(S) : NEXT
170 TA=&HFFFF-132*132+1  'SET TA TO START OF 17424-BYTE AREA
180 FOR Z=TA TO TA+132*132 : POKE Z,0 : NEXT
190 INPUT"LINE COORDS. (X1,Y1,X2,Y2)";X1,Y1,X2,Y2
200 IF X1=X2=Y1=Y2=-1 THEN 250
210 IF (X1<>X2 AND Y1<>Y2) OR (X1=X2 AND Y1=Y2) OR (X1>X2 OR Y1>Y2) OR (X1<0 OR
X1>131) OR (Y1<0 OR Y1>131) THEN 190
220 IF X1=X2 THEN 240
230 FOR N=X1 TO X2 : POKE TA+N+Y1*132,1 : NEXT N : GOTO 190
240 FOR N=Y1 TO Y2 : POKE TA+X1+N*132,1 : NEXT N : GOTO 190
250 LINE INPUT"LABEL:";A$
260 IF A$="" THEN 310
270 INPUT"LABEL COORDS. (X,Y)";X,Y
280 IF X<0 OR X>131 OR Y<0 OR Y>131 THEN 270
290 FOR N=1 TO LEN(A$) : POKE TA+Y*132+X+N-1,ASC(MID$(A$,N,1)) : NEXT
300 GOTO 250
310 'PRINT
320 FOR Y=0 TO 131
330 FOR X=0 TO 131
340 O=TA+X+Y*132
350 IF PEEK(O)=0 THEN LPRINT " "; ELSE IF PEEK(O)=1 THEN GOSUB 400 : LPRINT MID$
(S$,D,1); : ELSE LPRINT CHR$(PEEK(O));
360 NEXT X
370 LPRINT CHR$(27);CHR$(28)
380 NEXT Y
390 END
400 'SUB CHECK NEIGHBORS
410 D1=0 : D2=0 : D3=0 : D4=0
420 IF PEEK(O-132)=1 AND Y>0 THEN D1=1
430 IF PEEK(O+1)=1 AND X<131 THEN D2=1
440 IF PEEK(O+132)=1 AND Y<131 THEN D3=1
450 IF PEEK(O-1)=1 AND X>0 THEN D4=1
460 D=D1+D2*2+D3*4+D4*8
470 RETURN
```

Listing 10-1. Business Forms Program

A form produced by the program is shown in Figure 10-5. The form was designed by using a layout sheet, shown in Figure 10-6. On the layout sheet, the column positions are numbered from 0 through 131, and the lines are numbered from 0 through 131.

Number the intersections of lines as shown in Figure 10-5. Each intersection will have a column number, followed by a row number.

Now start the program. You should see the title

FORM PROGRAM

The program will next clear a large array that holds each print position. It will take several minutes to do this. Next, the program will ask:

Silicon Valley South Color Computer User's Group		
Name	Address	Phone
P.O. Box 9999, Silicon Valley South, CA		

Figure 10-5. Business Forms Program Example

LINE COORDS. (X1,Y1,X2,Y2)?

Enter the column, row coordinates of each line in the form. If the first line went from column 2, line 10 and ended at column 2, line 34, for example, you'd enter:

LINE COORDS. (X1,Y1,X2,Y2)? 2,10,2,34

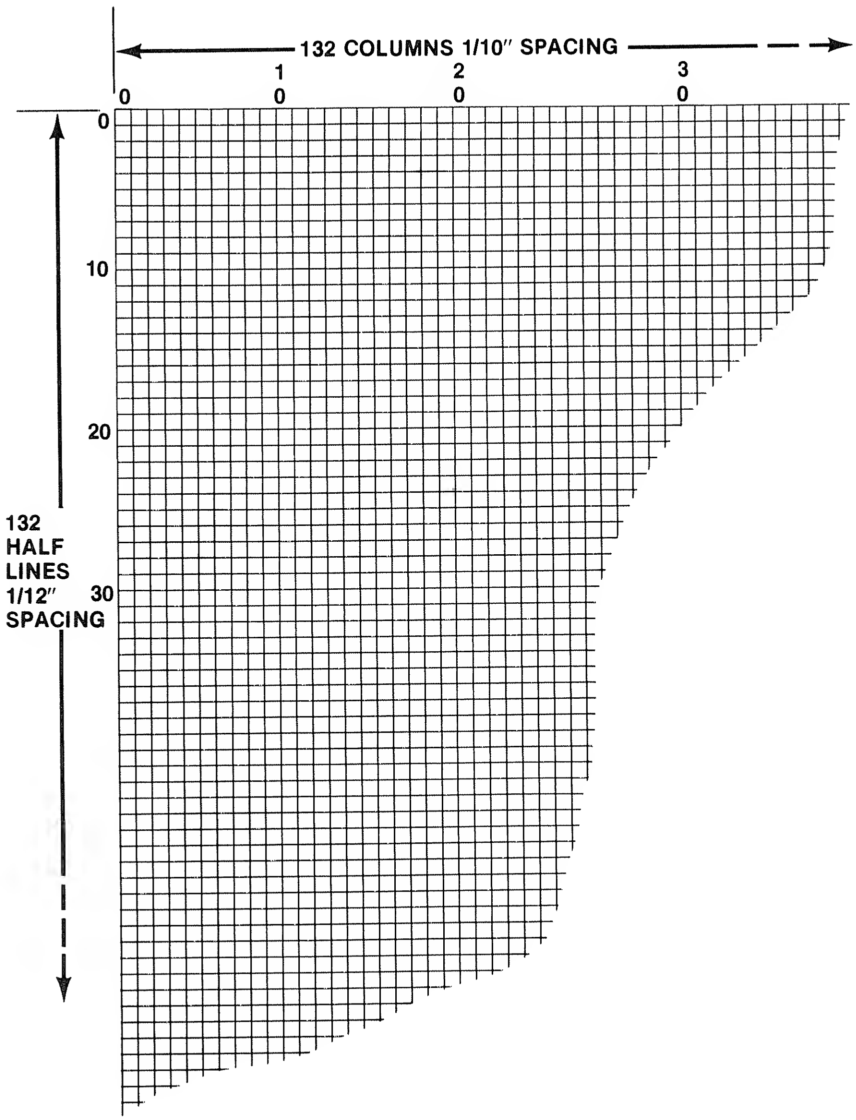


Figure 10-6. The Form Layout Sheet

Note that the first row is numbered 0 and the last row is numbered 131, and that the first column is numbered 0 and the last column is numbered 131. The coordinates prompt will be repeated—enter as many lines as there are lines on the form. It doesn't matter if you enter the line coordinates more than once. Always work from left to right and from top to bottom when specifying coordinates. The X1/Y1 coordinate should be the leftmost and topmost coordinate. The program will not accept coordinates entered with the X2/Y2 coordinate to the left or top of the X1/Y1 coordinate. When you've entered the last coordinate, enter a -1,-1,-1,-1 to tell the program that the entry is over.

Next, the program will ask you for labels for columns or other areas of the form. Enter the text for the label first. The program will then ask "LABEL

COORDS. (X,Y)?''. Enter the leftmost coordinate of the label. If you wanted "Part Number" to start at column 23 and row 5, for example, you'd enter

LABEL? Part Number

LABEL COORDS. (X,Y)? 23,5

Don't forget that each character in a label takes more than one half line and that you must space two half lines for normal text printing. Use an < ENTER > character alone to end the input once you've entered the last label.

After you've entered the last label, the program will start processing the coordinates. It constructs a matrix of print codes in an array for printing.

After you've entered the last label, the entire form will be printed out, complete with labels. You may have to get a printout several times before you've adjusted the final coordinates and labels, but this is the price you pay for not programming the form yourself!

Printer Hint

MORE ABOUT THE FORM PROGRAM

The form generator program works like this: First, the program reads the starting and ending points for the lines. It checks each set of points to make certain that the lines are either horizontal (X1 = X2) or vertical (Y1 = Y2), and that the points are within the form. For every character position along the line, a "1" is POKEd into the form array, contained in 132-by-132 bytes in RAM memory.

Next, the program reads the user text and coordinates, making similar checks on the starting point for the text. The text is also stored in the form array, in protected memory.

At this point, the array contains points and text. If a byte in the array is a "1," a line must be drawn. However, what type of line? To determine the type of line, the program checks the neighbors of each array-byte that contains a "1." Based upon the "line" or "no line" status of the neighbors, the proper block graphics character is selected. It's like the processing for the game of "Life" for those of you who are mathematical games freaks...

USING BLOCK GRAPHICS FOR DRAWINGS

The block graphics characters can also be used for drawing figures on printer paper. However, because the block graphics figures are rather coarse, you won't get the fineness of detail that's possible with full graphics. Each block graphics character represents one possible configuration for a two-by-two block matrix.

Each graphics element within the graphics block is about 1/4 the size of a normal character. As with the line segments, you'll have to set a half line feed by sending a CHR$(27);CHR$(28) to the printer. In 80 columns by 50 full-sized lines, the typical print area on an 8½- by 11-inch piece of paper, there are 160-by-100 graphics elements, or about 16,000 separately printable blocks, a graphics resolution that isn't too bad if you're printing a single picture on the page.

To print a picture using block graphics, first layout the picture on a layout form, as shown in Figure 10-7. The layout form will have 20 horizontal elements per inch and 12 vertical elements per inch. There will be four elements per character position. Black in the elements to be used and remember that you can use the triangles shown in Table 10-1. (The triangles take up four graphics elements, however.)

After blacking in the figure, convert the character positions to block graphics codes as shown in Figure 10-8. You can now construct a simple series of LPRINT statements that will print out the figure. Don't forget to initially set half lines:

```
100 LPRINT CHR$(27);CHR$(28)
110 LPRINT CHR$(XX);CHR$(XX); . . . CHR$(XX)
120 LPRINT CHR$(XX);CHR$(XX); . . . CHR$(XX)
   .
   .
   .
130 LPRINT CHR$(XX);CHR$(XX); . . . CHR$(XX)
```

The actual result is shown in Figure 10-9, along with an alternative program that reads DATA values to do the printing. The picture quality is good,

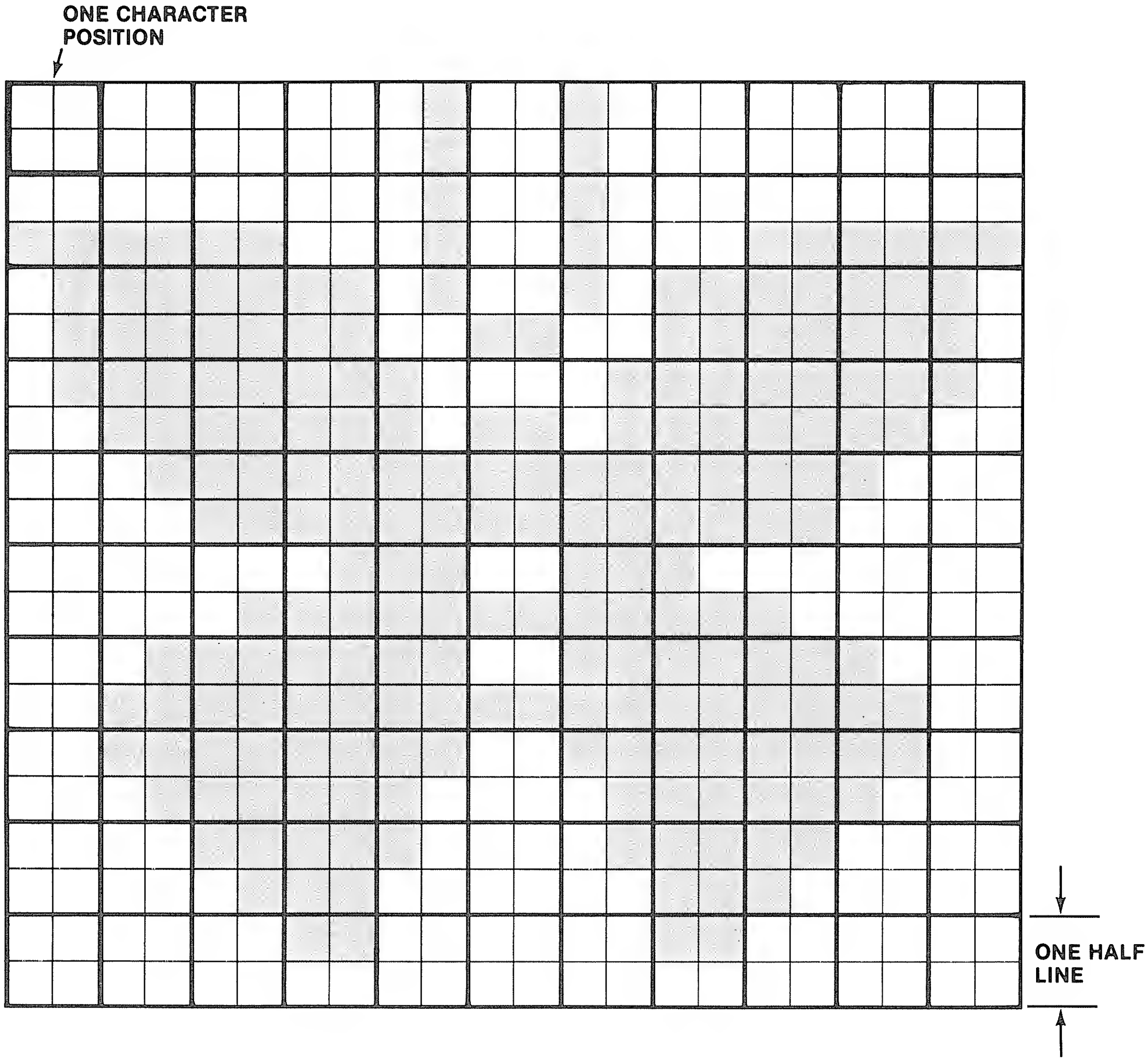


Figure 10-7. Graphic Layout Form for Block Characters

but not nearly as good as the standard graphics described elsewhere in this section, provided your printer is capable of the standard graphics. The block graphics can be used to black-in parts of forms, or for other functions, in addition to drawing pictures.

BASICS OF GRAPHICS MODE

Graphics mode is available in many newer Radio Shack dot-matrix printers. Printer graphics can be used to draw lines, to reproduce different character sets, or fonts, and to draw pictures. Virtually anything that can be broken down into dots can be reproduced with a dot-matrix printer. However, lest you think that it's simple to do this, we've got to add that while it's *possible* to produce incredible art using a dot-matrix printer, it may take a great deal of programming to accomplish the task.

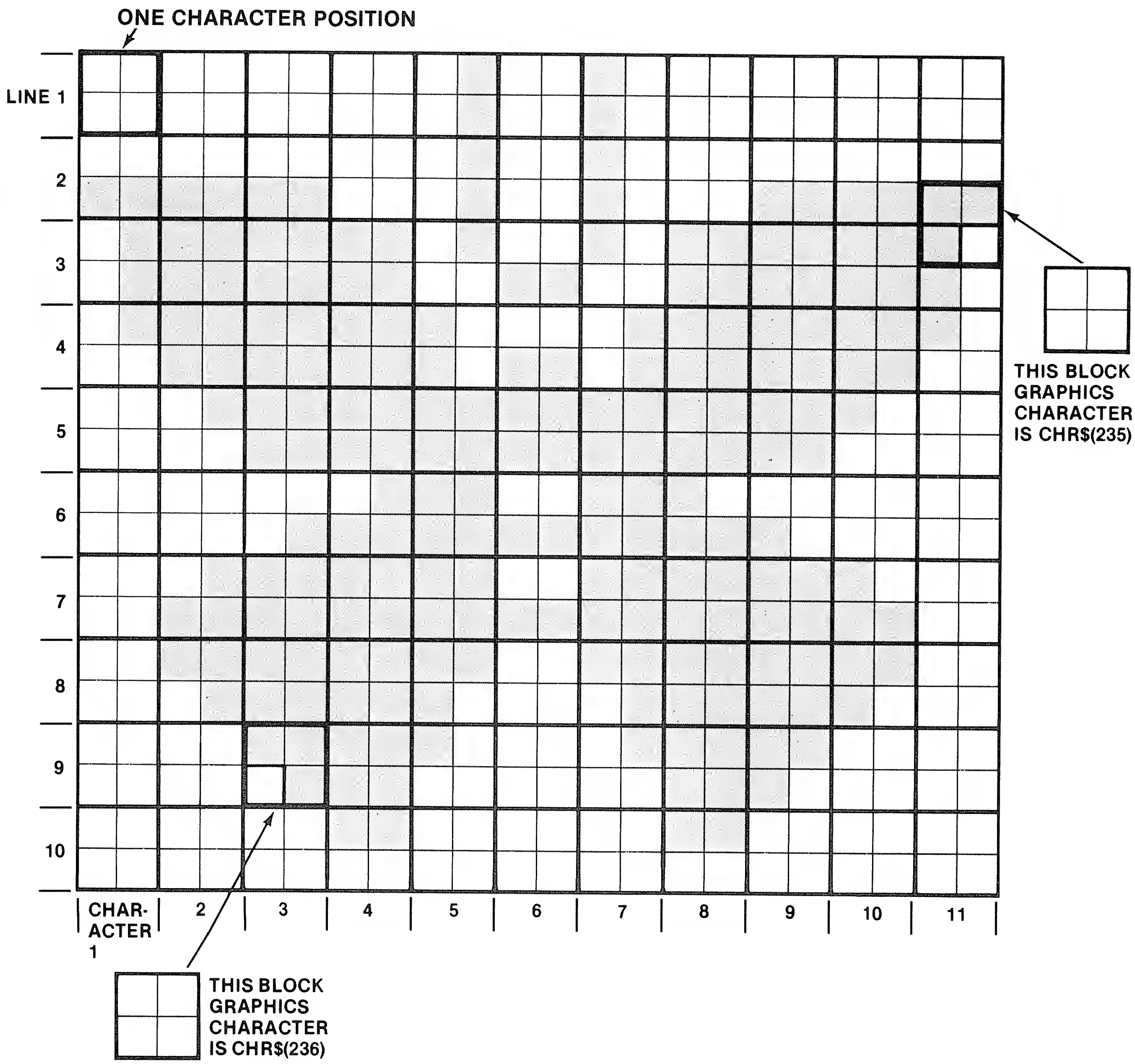


Figure 10-8. Converting to Block Graphic Codes

PRINTER GRAPHICS BASICS

All Radio Shack dot-matrix printers reproduce text characters by printing a dot-matrix representation of the characters.

When graphics mode is set in a dot-matrix printer, the character printing is disabled, and the print head is under control of the program (BASIC, assembly language, or another type). The program now passes data that tells

```
100 LPRINT CHR$(27);CHR$(28)
110 FOR I=1 TO 10
120 LPRINT
130 FOR J=1 TO 11
140 READ V
150 LPRINT CHR$(V);
160 NEXT J
170 NEXT I
180 LPRINT CHR$(27);CHR$(54)
190 'BUTTERFLY DATA VALUES
200 DATA 224,224,224,224,234,224,233,224,224,224,224
210 DATA 232,232,232,224,234,224,233,224,232,232,232
220 DATA 234,239,231,239,226,232,225,239,231,239,233
230 DATA 226,239,232,239,233,232,234,239,232,239,225
240 DATA 224,226,239,239,239,232,239,239,239,225,224
250 DATA 224,224,228,238,239,232,239,237,227,224,224
260 DATA 224,238,235,236,239,232,239,235,236,237,224
270 DATA 224,236,237,238,235,224,236,237,238,235,224
280 DATA 224,224,236,239,225,224,226,239,235,224,224
290 DATA 224,224,224,231,224,224,224,231,224,224,224
```

Figure 10-9. Sample Block Graphic Picture

the printer to *fire* one or more of the wires in the print head to print from one to seven dots in a vertical column. After the column is printed, the printer advances the print head to the next column. See Figure 10-10. The columns are generally the same columns that are used in printing text characters. Because each text character is made up of 6 to 20 columns, graphics printing controls 6 to 20 columns per *print position*, or 60 to 200 columns per horizontal inch, depending upon the printer involved.

The number of dots that print a vertical column is almost always seven in Radio Shack printers. (The DMP-110 prints 16 dots per column and the DMP-2100 prints 24 points per vertical column in a high-resolution mode, however. See "DMP-110 High-Resolution Graphics" and "DMP-2100 High-Resolution Graphics" in this section. This discussion applies to the low-resolution mode of the DMP-110 and DMP-2100).

Seven dots per vertical column is probably an offshoot of earlier Radio Shack printers which used seven vertical dots per character. Also, seven bits can conveniently fit into a byte to be sent to the printer. (The "most significant," or leftmost, bit was set in early printers to indicate a graphics character instead of a text character.)

The basic idea in graphics printing, then, is to print a row of seven-dot columns across the paper. The number of columns that can be printed varies with the printer and the pitch selected. A condensed or compressed pitch, of 12 or 16.7 characters per inch, operates with the dot columns closer together and if this spacing is maintained in graphics mode (it isn't always, depending upon the printer), you can fit more columns into the same width. The number of columns per inch and across the paper that can be held in each printer is shown in Table 10-2.

Printer Hint

SEVEN BITS GIVES PROGRAMMERS SLEEPLESS NIGHTS

Using seven dots per graphics column is a nuisance for most programmers who are so geared to powers of two that many have either 1, 2, 4, or 8 children. If graphics columns consisted of eight dots per column, then it would be much easier to process and store graphics data in memory and basic programs. As it is, you've got to divide a vertical coordinate by 7, which gives you the graphics line containing the dot. You've then got to equate the remainder of the division to one of seven dots, changing the remainder into a power of two. It sounds easy, but can get somewhat messy.

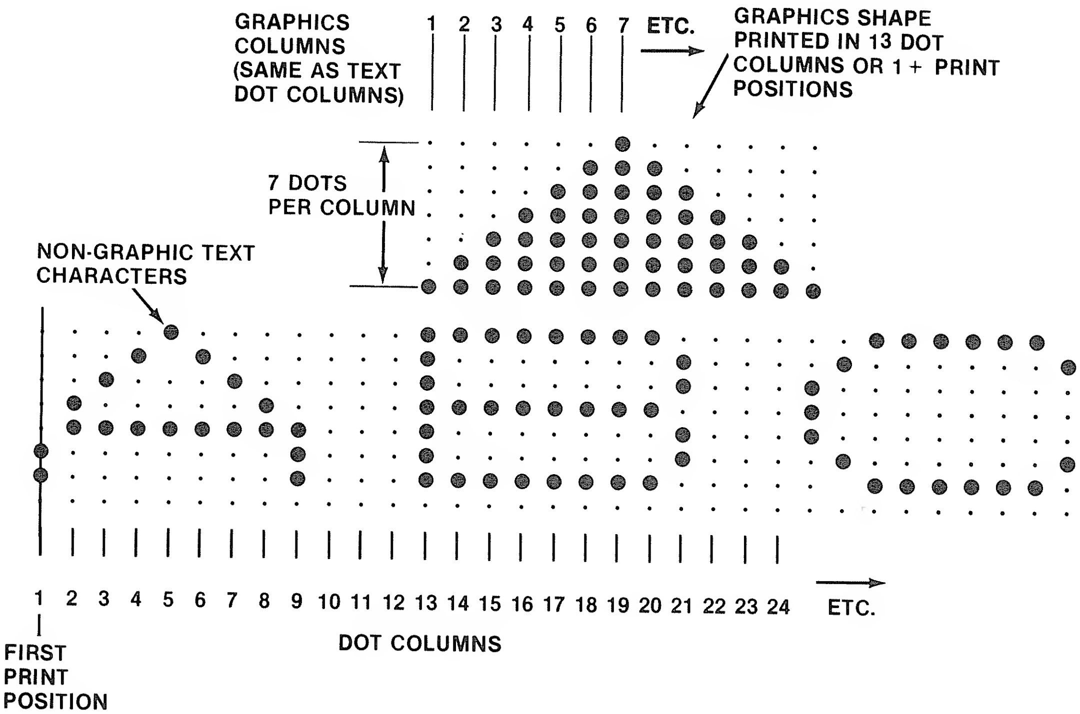


Figure 10-10. Column Printing in Graphics

Table 10-2. Printer column per inch spacing and dot columns

	# Dots/inch	# Dot addressable columns	Maximum width	Graphics line space	# Dots/ vertical inch
LPI:	N/A				
LPII:	N/A				
LPIII:	N/A				
LPIV:	N/A				
LPV:	N/A				
LPVI:	N/A				
LPVII:	60 fixed	480 fixed	8″	0.11″	63
LPVIII:	120 fixed	480 fixed[5]	8″	0.1″	72
DMP-100:	60 fixed	480 fixed	8″	0.11″	63
DMP-110:	120 fixed	960 fixed	8″	0.1″	72, 144[4]
DMP-120:	120[1], 200[3]	960[1], 1600[3]	8″	0.1″	72
DMP-200:	120[1], 144[2], 200[3]	960[1], 1152[2], 1600[3] All:[5]	8″	0.1″	72
DMP-400:	60[1], 72[2], 100[3]	792[1], 950[2], 1320[3] All:[5]	13.2″	0.1″	72
DMP-420:	60[1], 72[2], 100[3]	792[1], 950[2], 1320[3] All:[5]	13.2″	0.1″	72
DMP-500:	60[1], 72[2], 100[3]	792[1], 950[2], 1320[3] All:[5]	13.2″	0.1″	72
DMP-2100:	60[1], 180[4]	816[1], 2448[4]	13.6″	0.12″	60,180[4]
DW-I	N/A				
DW-II	N/A				
DW-IIB:	N/A				
DWP-210:	N/A				
DWP-410:	N/A				
CGP-115:	N/A				
CGP-220:	80 fixed	640 fixed	8″	0.12″	60
QP-I:	N/A				
QP-II:	N/A				
TP-10:	N/A – block graphics only				
Plotter/ Printer	N/A				

N/A = No graphics [1] = Standard [2] = Compressed [3] = Condensed [4] = High resolution [5] = Two dots printed for every dot column

If the printer were to keep the same spacing for graphics lines as for text lines, there'd be a gap between adjacent rows, as shown in Figure 10-11. Since we want to be able to print all dot positions on the paper, the graphics mode on all printers operates with a *graphics mode line spacing* that butts one graphics row against the preceding graphics row, as shown in the figure. This line spacing is again dependent upon the printer, but is about 1/10 of an inch, or a line spacing of ten graphics lines per inch, rather than the nominal six lines per inch used in a lot of text printing.

BASIC allows you to address a dot on a graphics screen by using a SET (PSET) or RESET (PRESET) command that uses x,y coordinates. However, there is no corresponding BASIC command for printing a graphics page on a printer. It would be nice to have a built-in BASIC command that would allow you to print the 100th dot in the 16th row of the page. However, the command is just not available. How, then, do you specify the dots to print when using printer graphics? You have to know the pattern or image you wish to print beforehand, and convert it into seven-dot vertical columns, dot column numbers, and graphics lines.

A SIMPLE EXAMPLE OF GRAPHICS PRINTING

Suppose that you want to draw a triangle, five lines down and 402 dot columns over, as shown in Figure 10-12. The program below shows how it would be done. The steps to follow are these:

1. Set graphics mode.
2. Space a number of graphics lines down.
3. Space over a number of dot-columns.
4. Print the required columns, one column at a time.
5. Repeat for the next graphics to be drawn.

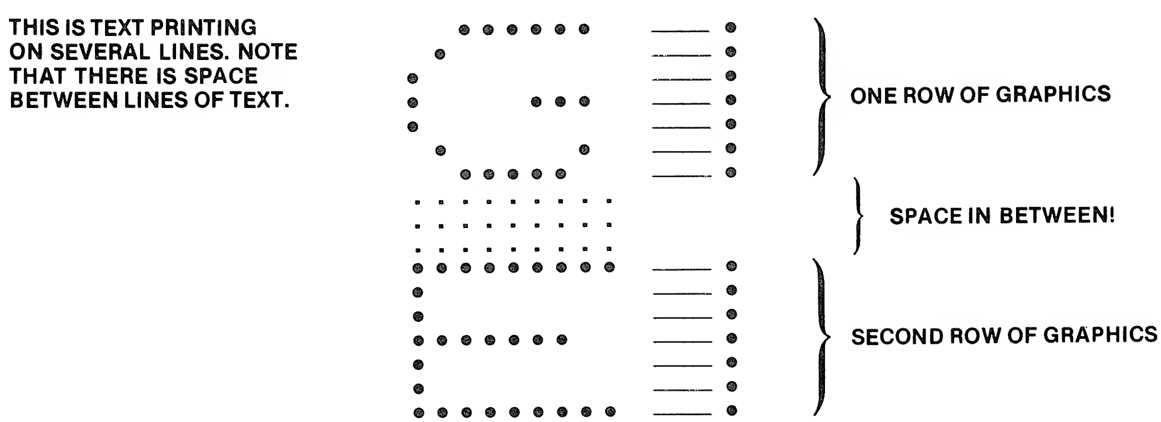


Figure 10-11. Improper Graphics Row Spacing

```
100 LPRINT CHR$(18);
110 LPRINT: LPRINT: LPRINT: LPRINT: LPRINT
120 LPRINT CHR$(27);CHR$(16);CHR$(1);CHR$(146)
130 LPRINT CHR$(192);CHR$(224);CHR$(208);CHR$(200);
140 LPRINT CHR$(196);CHR$(194);CHR$(193);CHR$(194);
150 LPRINT CHR$(196);CHR$(200);CHR$(208);CHR$(224);
160 LPRINT CHR$(192)
```

Let's look at those steps in detail. First, graphics mode must be set. Graphics mode is almost always set by sending a CHR$(18) to the printer. From the time that the printer receives the CHR$(18) until graphics mode is reset by a CHR$(30), the printer will assume that it's receiving data that defines graphics dot column positions or dots to be printed.

Next, the required number of graphics lines are skipped. Graphics lines are skipped the same way you'd skip text lines—by sending a carriage return by an LPRINT. In this case, the printer recognizes that graphics mode is set and instead of doing a full line space, does a graphics-mode line space, about 1/10 of an inch. In the program here, we've issued five LPRINTs to space down five lines.

We're now at the proper graphics line. The next step is to space over the required number of dot columns. Consult Table 10-2 for the number of dot columns your printer uses. We'll assume that you haven't set a pitch other than the standard 10-pitch (ten characters per inch) that is the default condition for the printer (dependent upon the DIP switch settings in the Model 2100). The printer knows where the print head is located at all times. Sending this character string will move the print head to the proper dot column:

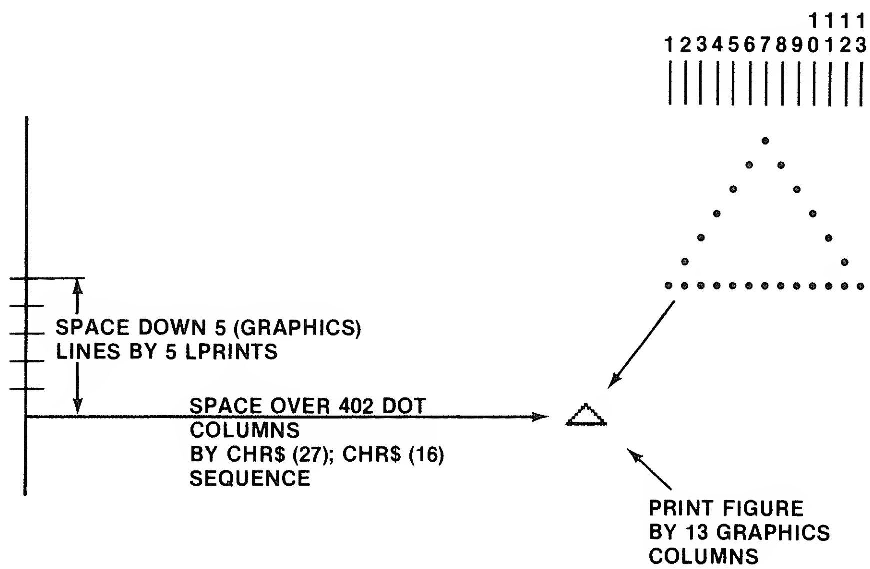


Figure 10-12. Sample Graphics Figure

CHR$(27);CHR$(16);CHR$(MM);CHR$(LL)

The only sticky problem here is in the MM and LL parameters. Taken together, they represent the dot column for the print head position. To find the proper values, divide the dot-column position by 256, and save the remainder. The quotient is MM and the remainder is LL. In this case we want to position the print head to dot column 402. Dividing 402 by 256 gives the answer, 1 with a remainder of 146. MM is therefore 1 and LL is 146.

Note: Because this escape sequence contains numeric values which may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, identical to the top-of-form character, the print driver will attempt to execute a page-eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line feed character, it will be changed to a 13 in the print driver. Avoid positioning commands containing these values by using a slightly larger or smaller value in the CHR$() numeric variable, or use a combination of two or more positioning commands.

We're now positioned at the proper dot column. The triangle figure is made up of 13 columns. We'll have to send 13 CHR$() values to fire the print head at 13 columns. Before you can send the values, however, you'll have to *encode* the dot patterns into the CHR$() values. To do this, follow the example in Figure 10-13. The topmost dot has a *weight* of 1, the next 2, the next 4, the next 8, the next 16, the next 32, and the next 64. For every dot that has to be printed, add its value into the CHR$() total and *then add 128*. In the third column, for example, we have the fifth and seventh dot, so we'd add 16 and 32 for the two dots plus 128, for a CHR$() value of 176. The 128 value marks the data as graphics data rather than a text character.

After the last column is printed, the print head is located over the next dot column, ready for the next command.

This is the general procedure for printing any graphics character or block of characters. It's tedious, but necessary for printing graphics.

For more on graphics see "Printing Pictures in Graphics Mode."

REPEAT CODES FOR GRAPHICS

Repeat codes for graphics are used in the same way as for text—to repeat a character more than one time. It's a condensed way of instructing the printer to print identical graphics characters, and is similar to the STRING$ function in BASIC. Repeat codes are available in the LPVII, LPVIII, and all "DMP" printers except the DMP-120, CGP-220, and TP-10. Suppose that you were printing a double line with a graphics character, with both the top and bottom dots set for 200 dot columns, as shown in Figure 10-14,

You could send a whole series of graphics characters:

```
100 LPRINT CHR$(18);
110 LPRINT CHR$(193);CHR$(193);CHR$(193) . . .
```

Printer Hint

USING THE DOT-COLUMN VALUE

The dot-column value is really a 16-bit binary number. If you don't know anything about binary, you can skip this explanation and get along nicely. However, if you'd like to know more about the format, read on.

The dot column is expressed in two bytes, or 16-bits. Unlike 16-bit values in the Model I, II, III, and 4, (but like 16-bit values in the Color Computer and MC-10) this 16-bit number is ordered *most significant byte* followed by *least significant byte*. Values up to 65,535 can be held in 16 bits, so there's really no problem in specifying dot column numbers here (at least until we get the next generation of printers!).

Printer Hint

HOW TO FIGURE THE DOT VALUES

The dot values for each dot column are really an eight-bit binary number. (If you don't like binary, skip this section.) You can look at the "weights" as the 1 bits in a binary value. The bottommost dot is the bit in bit-position six, the next dot is the bit in bit-position five, and so forth, up to the top dot, which is the bit in bit-position zero. The highest-order bit, bit-position seven, is always set to signify graphics data. If you were printing (from the bottom of the column), dot, no dot, dot, no dot, dot, no dot, dot, you'd have 11010101 for the value, or decimal 213, the same as 128 + 64 + 0 + 16 + 0 + 4 + 0 + 1.

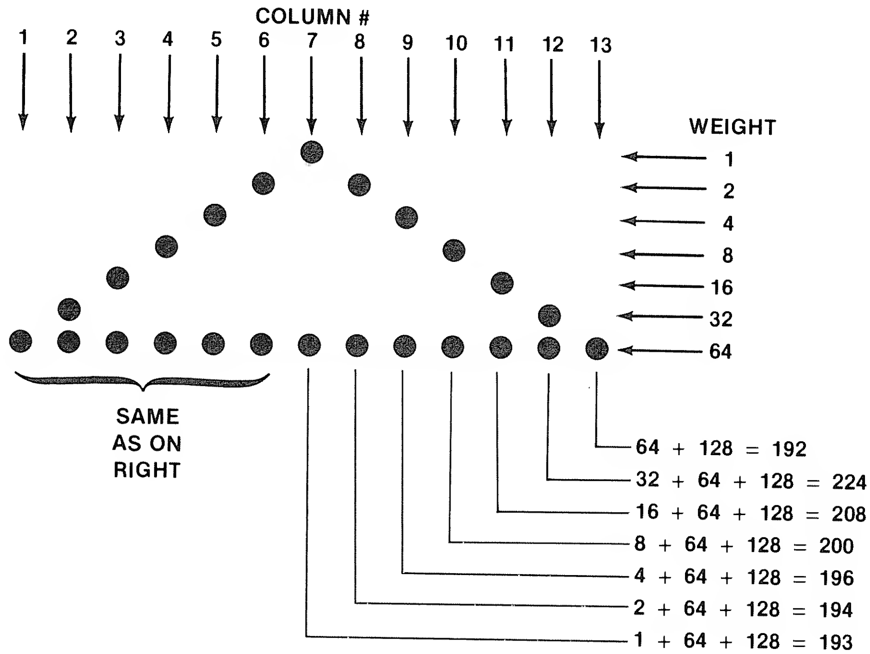


Figure 10-13. Coding Graphics Columns

or you could do a loop

```
100 LPRINT CHR$(18);
110 FOR I=1 TO 200
120 LPRINT CHR$(193);
130 NEXT I
```

You could also use a BASIC STRING$ command (as discussed in a Printer Hint in Section II). However, another way to do the task is to use a repeat code. The repeat function uses the special escape sequence CHR$(28);CHR$(NN);CHR$(CC), where NN is the number of times that a character will be repeated, and CC is the value of the character to be repeated. We could have used CHR$(28);CHR$(200);CHR$(193) above.

The NN value can be 0 through 255. (A value greater than 255 will not be accepted by BASIC, and if sent to the printer would not be handled properly. The printer expects to see the repeat count in the next character and values greater than 255 would have to be in two bytes or characters. If a 0 value is accepted by the printer it will be treated as a 256, however.)

The repeat function is useful in graphics applications because the printing density is so much greater than text printing, and it would be tedious to use a discrete CHR$() value for every graphics character to be printed, as would be the case in sending a CHR$(193) 200 times above.

Note: Because the repeat escape sequence contains numeric values that may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, identical to the top-of-form character, the print driver will attempt to execute a page eject by doing a series of new lines after it receives the 12. If a

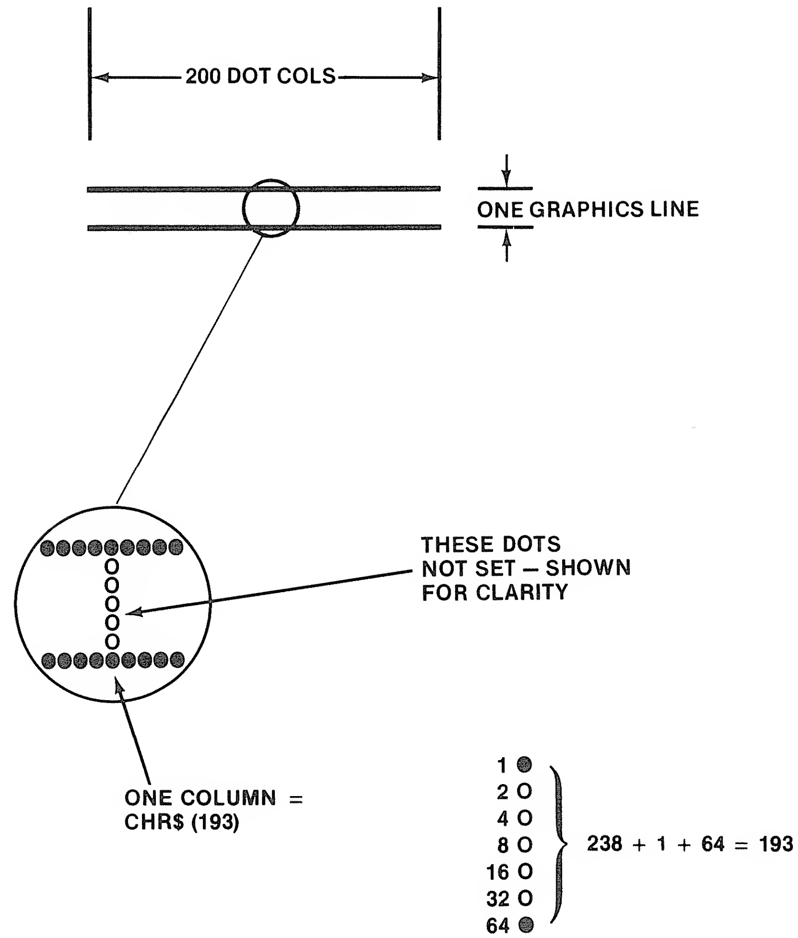


Figure 10-14. Repeat Code Example

numeric value turns out to be a 10, identical to a line-feed character, it will be changed to a 13 in the print driver. Avoid repeat sequences containing these values by using a slightly larger or smaller value in the CHR$() numeric variable, or use a combination of two or more repeats to get the same result.

Another good alternative is to use the STRING$ function in BASIC to repeat from within a BASIC statement. If you use STRING$, however, beware of sending *too many* characters to the printer through the printer driver! Too many characters may cause the printer driver to interject a line feed when it thinks that the number of characters per line has been exceeded. One solution to this is to periodically reset the count of the number of characters for the current line in the BASIC driver (location 155 in the Color Computer, location 16426 in the Model III and 4). This fools the print driver into thinking that the number of characters in the current line is less than the number allowed, and prevents a spurious line feed.

CHAPTER 11

SCREEN PRINTING

In this chapter we'll tell you how to use your printer to print a graphics screen. Graphics on the Model I, III, and 4 is quite a bit different than on the Color Computer and MC-10, so we'll have to split up this discussion into two parts, each one based upon the type of computer system. In addition to differences between computer systems, there are differences in the approaches to be used.

GRAPHICS ON THE MODEL I, III, AND 4

Before we discuss how to print the screen on the Model I, III, and 4, let's refresh our memory about what graphics are available on these systems. Screen memory on the three systems starts at location 15360 through 16383, as shown in Figure 11-1. There are 64 columns per line and 16 lines on the screen. Each column represents one character position—a single text or graphics character can be held in that character position. There are 1024 character positions on the total screen.

PRINTing text on the screen is no problem. Characters are displayed by doing a PRINT, which will display text at the next line on the screen; the screen will scroll as more lines of text are written. PRINT @ can be used to PRINT text starting at any location on the screen—the location number from 0 through 1023 is used in the PRINT @ statement. PRINT @128,"TEXT", for example, prints TEXT at location 128, the start of the third line. Text can also be printed by a POKE command directly to one of the screen memory locations. When a POKE is done, an ASCII character is stored directly in screen memory.

Each of the 1024 character positions can be used to hold a graphics character instead of a text character. Graphics characters look like Figure 11-2. There are six graphics elements per character, arranged in two columns by three rows. A graphics element can be either off (black) or on (white). Because there are six graphics elements per character position, there is a total of 6144 (1024 times 6) graphics elements on the screen, arranged in columns numbered 0 through 127 and rows numbered 0 through 47.

A graphics element is normally set by the SET command, which sets (turns on) a specified graphics element at a given column, row position. Doing a SET (64,8), for example, will set the graphics element in the middle of line 4. A graphics element can be reset (turned off), by doing a RESET com-

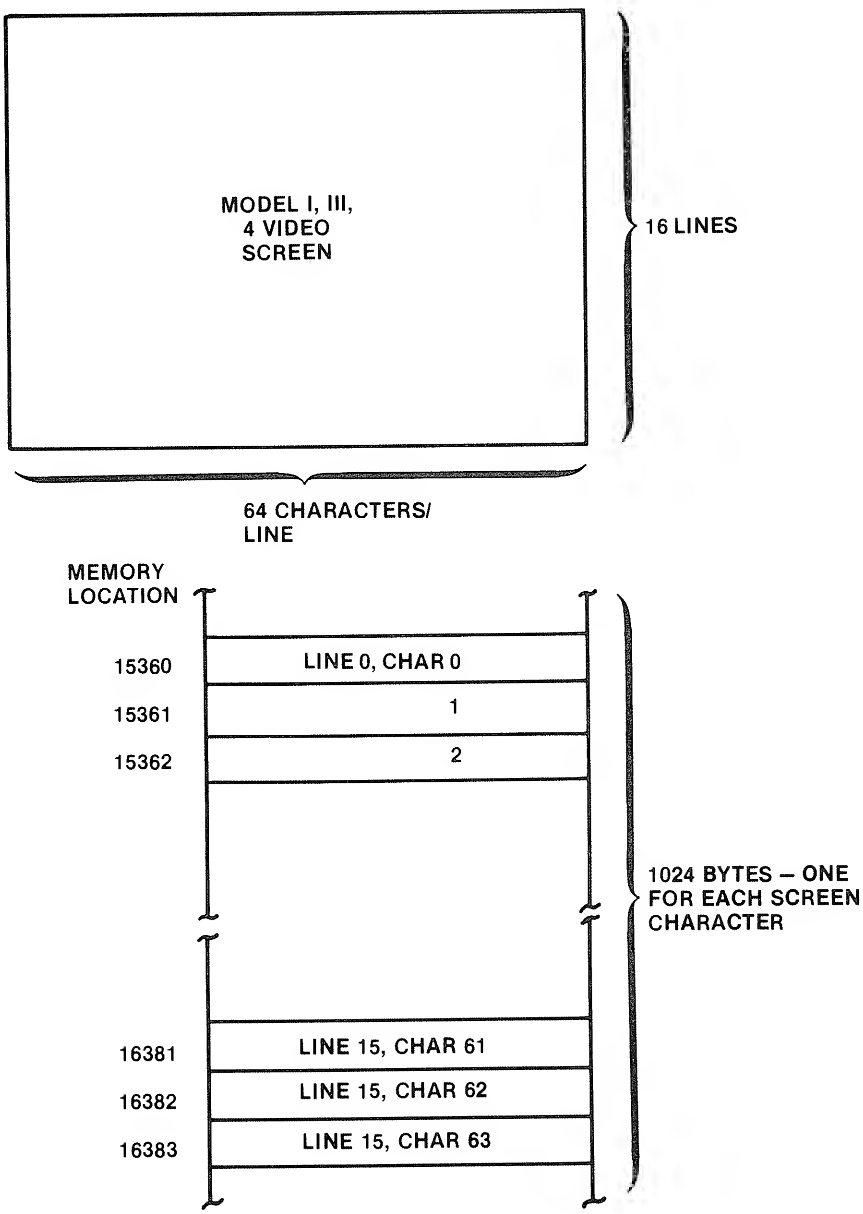


Figure 11-1. Model I, III, 4 Screen Memory Map

mand with the same format. A POINT command reads back the contents of the graphics element—A = POINT(63,31), for example, sets variable A to -1 if the graphics element is on, or to a 0 when the graphics element is off.

Graphics elements can also be set by using POKEs. In this case, the POKE address is to one of the screen locations 15360 through 16383. The value for the POKE is calculated by assigning weights to each of the elements in a character position, as shown in Figure 11-3. The weights of 1, 2, 4, 8, 16, and 32 are added together to get the value for the POKE. A value of 128 is added to the result to indicate to the system that the character position holds a graphics character and not a text character.

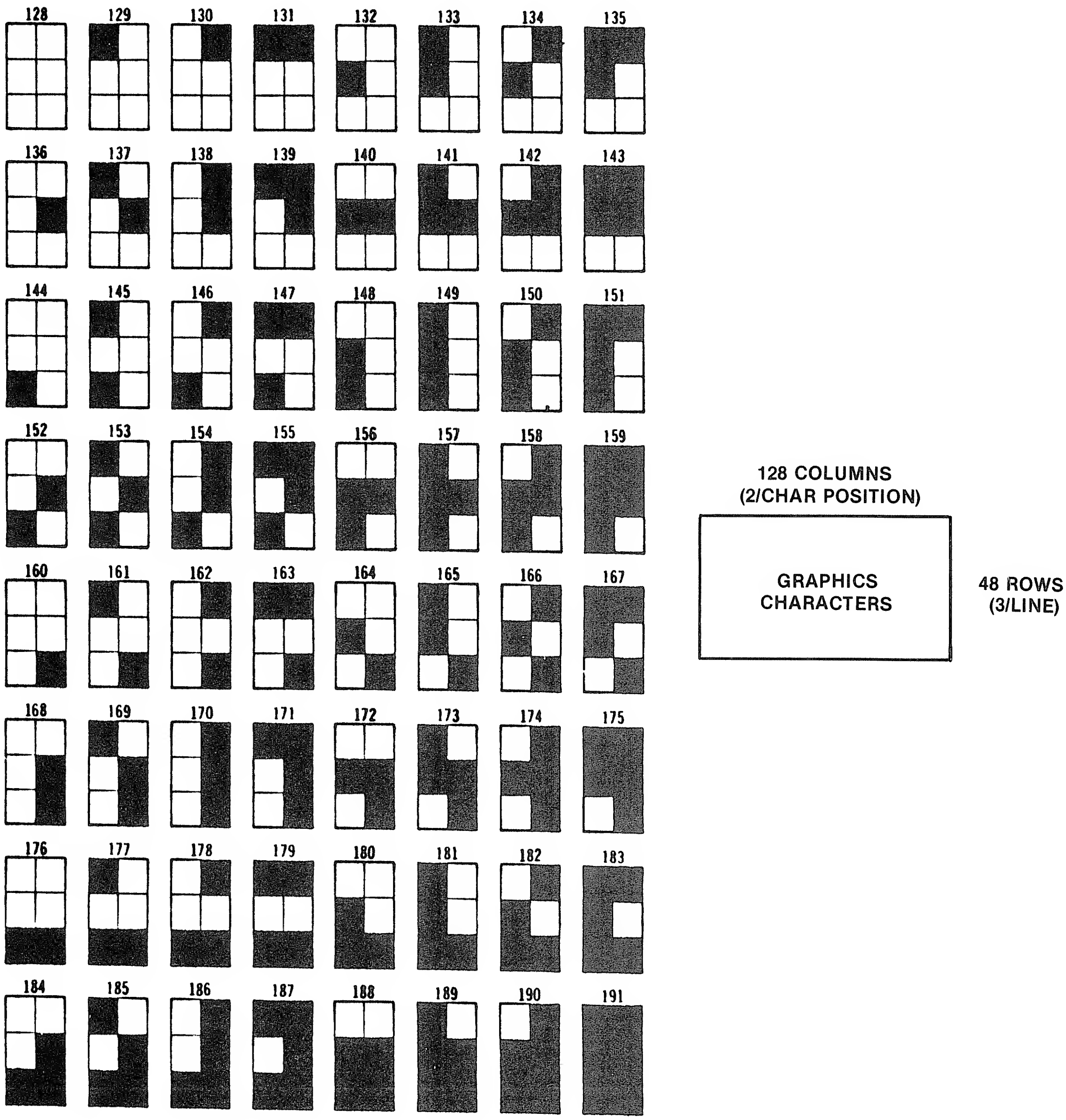


Figure 11-2. Model I, III, 4 Graphics Characters

Another method for setting graphics elements is to use a CHR$ function in a string. The value for the CHR$ is calculated, as in the POKE case, and used in the graphics string. The string "THIS IS TEXT " + CHR$(153) + " AND GRAPHICS" results in the intermixed text and graphics shown in Figure 11-4.

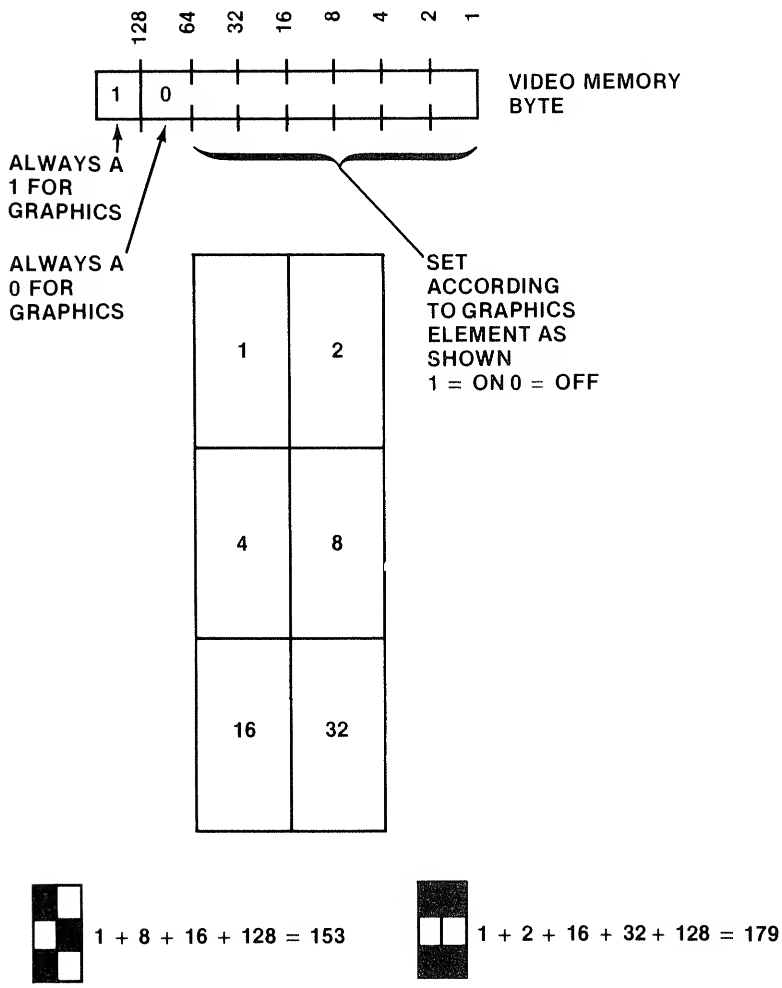


Figure 11-3. Model I, III, 4 Graphic Character Weights

GENERAL SCHEME FOR PRINTING GRAPHICS

The general approach to printing graphics from the screen is to read all of the 1024 screen locations from video memory at 15360 through 16383 directly by using PEEKs. The PEEK command reads the contents of a memory location. The program can then decode the graphics value from the PEEK variable and print out a corresponding block graphics or high-resolution graphics character to reproduce the screen.

One of the biggest problems in dumping the screen is achieving the same proportions on the printout as appear on the screen. The Model I, III, and 4 screen has an aspect ratio of about 4-to-3—four horizontal units to three vertical units. This makes a graphics element about 4/128 or 1/32 of a unit horizontally by 3/48 or 1/16 of a unit vertically, roughly a 1-to-2 aspect ratio for each graphics element. To reproduce the screen on a printer, the program must have the same proportions on the printout of graphics elements.

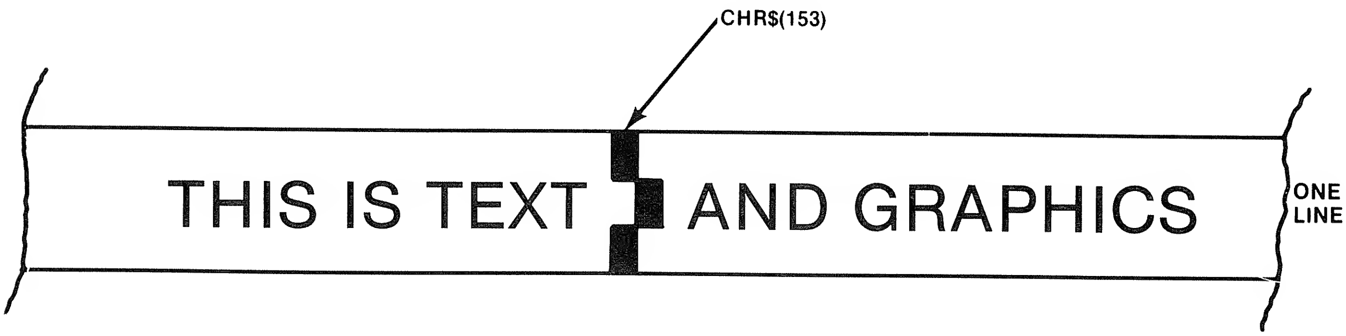


Figure 11-4. Intermixed Test and Graphics on the Model I, III, 4

USING BLOCK GRAPHICS TO PRINT OUT THE SCREEN

Block graphics are discussed in "Using Block Graphics" in this section, so you might want to read that material to get a background on the block graphics characters. Block graphics are available in the newer printers and are really a subset of characters in the range 224 to 254. They are printed not in graphics mode, but in normal text mode.

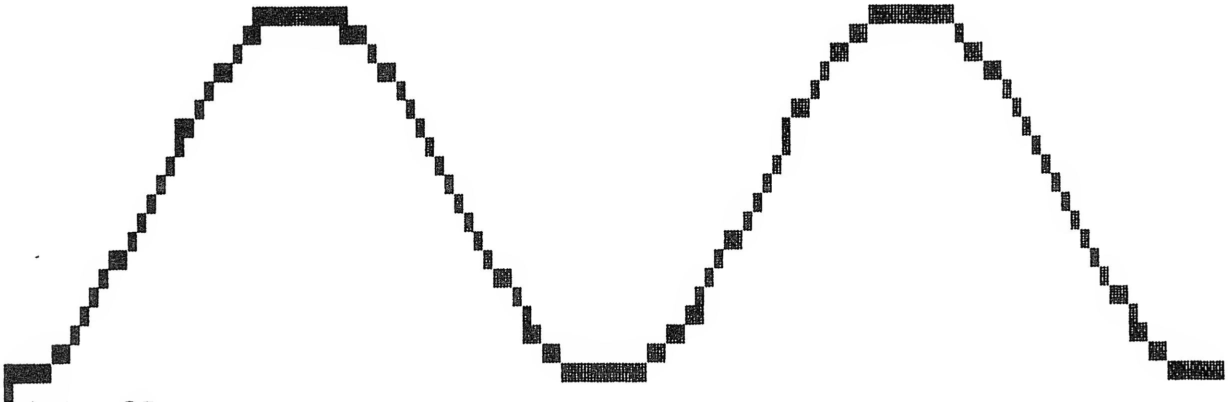
Elements in the block graphics characters print at about 1/20 of an inch horizontally by 1/24 of an inch vertically. That's an aspect ratio of (1/20):(1/24), or about 24/20, or 6 to 5. However, if two vertical block graphics elements are used for every screen element, the ratio becomes 3 to 5. Although the 3 to 5 ratio is not equal to the aspect ratio of the screen, it's not a bad match to the screen graphics element ratio of one to two.

The program in Listing 11-1 uses block graphics characters to print a screen made up of graphics and text characters. The program is a subroutine that you can call from your own BASIC program or simply use as a stand-alone program. (To use the program as a stand alone program, delete the RETURN line.) Figure 11-5 shows a typical printout.

```
10000 'SUB MODEL I,III,4 BLOCK GRAPHICS SCREEN PRINT
10010 CH$(0)=CHR$(224) : CH$(1)=CHR$(233) : CH$(2)=CHR$(234) : CH$(3)=CHR$(239)
10020 LPRINT CHR$(27);CHR$(28); 'SET HALF LINE FEED
10030 FOR ZY=0 TO 15
10040 FOR ZX=0 TO 63
10050 ZP=15360+ZX+ZY*64
10060 IF PEEK(ZP)<128 THEN ZZ$(1)=ZZ$(1)+" " : ZZ$(2)=ZZ$(2)+CHR$(PEEK(ZP)) : ZZ
$(3)=ZZ$(3)+" " : GOTO 10080
10070 ZZ$(1)=ZZ$(1)+CH$(PEEK(ZP) AND 3) : ZZ$(2)=ZZ$(2)+CH$((PEEK(ZP) AND 12)/4)
 : ZZ$(3)=ZZ$(3)+CH$((PEEK(ZP) AND 48)/16)
10080 NEXT ZX
10090 FOR ZX=1 TO 3 : LPRINT ZZ$(ZX) : ZZ$(ZX)="" : NEXT ZX
10100 NEXT ZY
10110 LPRINT CHR$(27);CHR$(54) 'RESET FULL LINE FEED
10120 RETURN
```

Listing 11-1. Model I, III, 4 Block Graphics Screen Print Program

Cosine Curve



```
ist 120
120 FOR X=0 TO 12.7 STEP .09 : SET(X*10,COS(X)*10+20) : NEXT
130 PRINT @ 90,"Cosine Curve"
READY
>run
```

Figure 11-5. Model I, III, 4 Block Graphics Screen Print Example

USING STANDARD GRAPHICS TO PRINT THE SCREEN

Standard graphics can also be used to print the screen of the Model I, III, or 4. The general subject of how to print standard graphics is covered in "Basics of Graphics Mode," in this section. You might want to read that material if you're unsure about how to print in standard graphics.

We could use standard graphics to print only graphics characters. However, it might be overkill and it might be wiser to use the less complicated block graphics for this task. Graphics printing *can* be used to print both text and graphics characters on the Model I, III, 4, and 4P screen. Here's how.

The video display of these systems is made up of 1024 character positions, as noted previously. Each character position is divided into 6 horizontal by 12 vertical *pixels*, or picture elements, as shown in Figure 11-6. Text characters use a five-by-seven dot-matrix for uppercase, and a five-by-eight dot-matrix for lowercase with descenders, as shown in the figure. The remaining four or five rows at the bottom of the character position are blank, to give a separation between lines. The righthand column for every text character position is also blank, to provide space between text characters.

Graphics characters use a three-by-four matrix for each of the six graphics elements, as shown in Figure 11-7.

Radio Shack printer high-resolution graphics is done by printing columns of seven dots, as described in "Basics of Graphics Mode." The problem in doing a screen print on the Model I, III, 4, or 4P is using the seven-dot column to

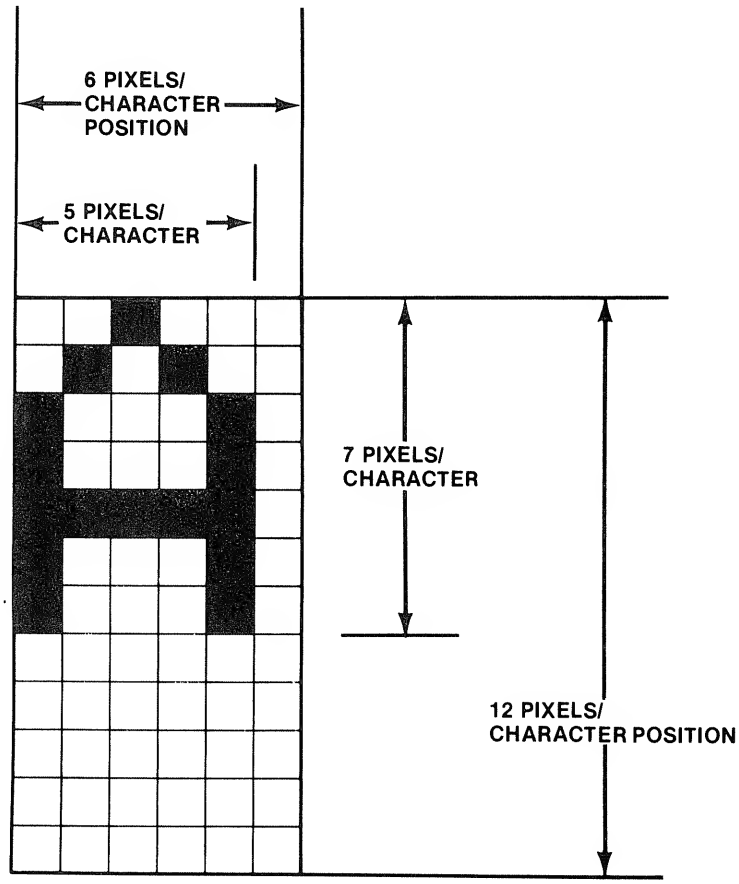


Figure 11-6. Model I, III, 4 Character Pixels

Printer Hint

HOW THE MODEL I, III, AND 4 BLOCK GRAPHICS SCREEN PRINT PROGRAM WORKS

This program scans the video memory at RAM locations 15360 through 16383 by a PEEK from each location. The video memory is scanned a row at a time. For each row, three row segments are constructed and held in ZZ$(1) through ZZ$(3), a three-element string array. If the current character is displayable, it is added to the ZZ$(2) string and blanks are added to ZZ$(1) and ZZ$(3). If the current character is a graphics character, the proper block graphics character is added to ZZ$(1), ZZ$(2), and ZZ$(3). At the end of each row, ZZ$(1), ZZ$(2), and ZZ$(3) are printed out in three half-line feed lines. As each ZZ$ string is printed, it is set to a "null" value in preparation for the next line.

reproduce as accurately as possible the 192 pixels represented in 16 vertical columns. Printing one graphics column for a character position won't work, because a graphics column is seven dots, and a minimum of 12 is required for both text and graphics characters. Printing two graphics columns might work because that would be close to the 12 dots required, but 14 can't be divided evenly by three, to separate the column into graphics elements. Printing three graphics columns per character position appears to be the answer—that's 21 dots, or seven dots per graphics element. The 21 dots can be divided into 14 or 16 dots for character representation—not completely accurate, but close enough. See Figure 11-8.

Standard printer graphics uses about 0.1- to 0.12-inch line spacing. If we print three graphics columns for every character position, we'll have 16 times 3 = 48 lines at 12 lines per inch, or 4.8 to 5.8 inches vertically to represent the height of the screen. What about the horizontal dimension, ideally at about 6.4 to 7.7 inches, to keep the screen aspect ratio of 4 to 3? The width of the screen print will depend upon the print position density along the line. Chances are your printer can use variable pitch, but assume a standard pitch of ten characters per inch. The dot column density is usually about 60 columns per horizontal inch, or six columns per character position. To print six horizontal pixels in each character position, we need 64

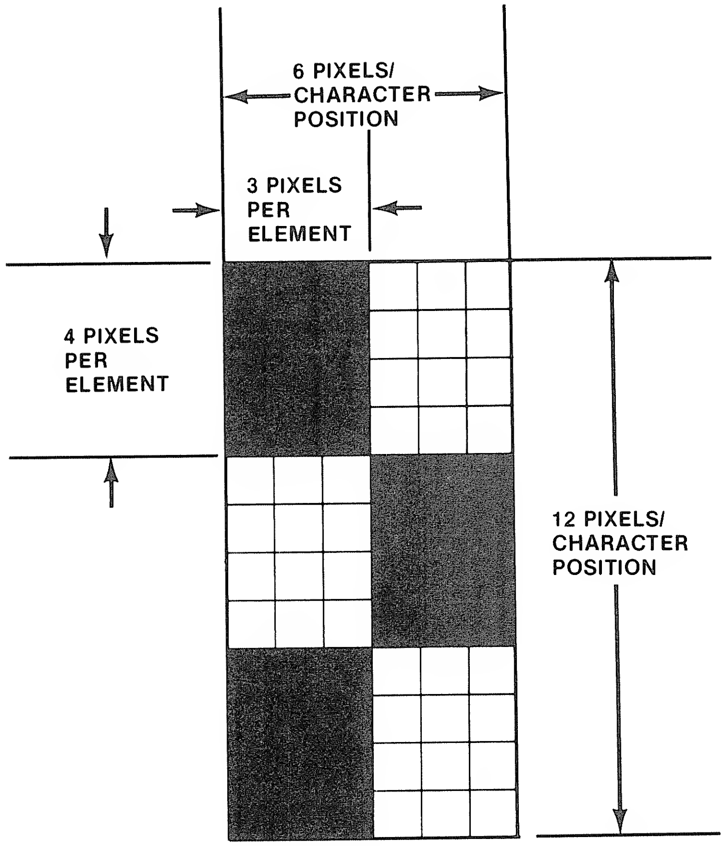


Figure 11-7. Model I, III, 4 Graphics Pixels

times 6, or 384 graphics columns. At 60 columns per inch, that would be about 6.4 inches, close to the proportion for the screen, and still narrow enough to fit on a standard paper width. Your printer may have somewhat different proportions in graphics for the screen width.

A GRAPHICS SCREEN PRINT

The program in Listing 11-2 is a graphics screen print program that will print both text and graphics from a Model I, III, 4, or 4P screen. It can be used as a subroutine in your own BASIC program, or can simply stand alone by deleting the RETURN line. Figure 11-9 shows the printout from the program for a typical screen.

GRAPHICS ON THE COLOR COMPUTER AND MC-10

There are two types of graphics available on the Color Computer, and one currently available on the MC-10. Low-resolution text graphics is available on both machines. High-resolution graphics is possible on the MC-10, but currently is only supported by the Color Computer.

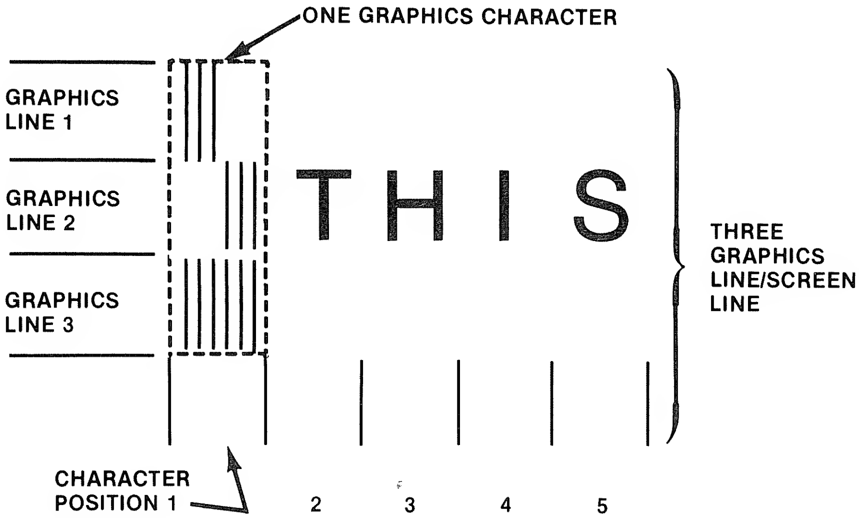


Figure 11-8. Model I, III, 4 Standard Graphics Program Format

```
10000 'SUB MODEL I,III,4 STANDARD GRAPHICS SCREEN PRINT
10010 CH$(0)=" "
10020 CH$(1)=CHR$(18)+CHR$(28)+CHR$(3)+CHR$(255)+CHR$(28)+CHR$(3)+CHR$(128)+CHR$
(30)
10030 CH$(2)=CHR$(18)+CHR$(28)+CHR$(3)+CHR$(128)+CHR$(28)+CHR$(3)+CHR$(255)+CHR$
(30)
10040 CH$(3)=CHR$(18)+CHR$(28)+CHR$(6)+CHR$(255)+CHR$(30)
10050 FOR ZY=0 TO 15
10060 FOR ZX=0 TO 63
10070 ZP=15360+ZX+ZY*64
10080 IF PEEK(ZP)<128 THEN ZZ$(1)=ZZ$(1)+" " : ZZ$(2)=ZZ$(2)+CHR$(PEEK(ZP)) : ZZ
$(3)=ZZ$(3)+" " : GOTO 10100
10090 ZZ$(1)=ZZ$(1)+CHR$(PEEK(ZP) AND 3) : ZZ$(2)=ZZ$(2)+CHR$((PEEK(ZP) AND 12)/
4) : ZZ$(3)=ZZ$(3)+CHR$((PEEK(ZP) AND 48)/16)
10100 NEXT ZX
10110 FOR ZX=1 TO 3
10120 FOR ZZ=1 TO LEN(ZZ$(ZX)) : IF ASC(MID$(ZZ$(ZX),ZZ,1))>3 THEN LPRINT MID$(Z
Z$(ZX),ZZ,1); ELSE LPRINT CH$(ASC(MID$(ZZ$(ZX),ZZ,1)));
10130 NEXT ZZ : LPRINT CHR$(18) : LPRINT CHR$(30); : ZZ$(ZX)=""
10140 NEXT ZX,ZY
10150 RETURN
```

Listing 11-2. Model I, III, 4 Standard Graphics Screen Print Program

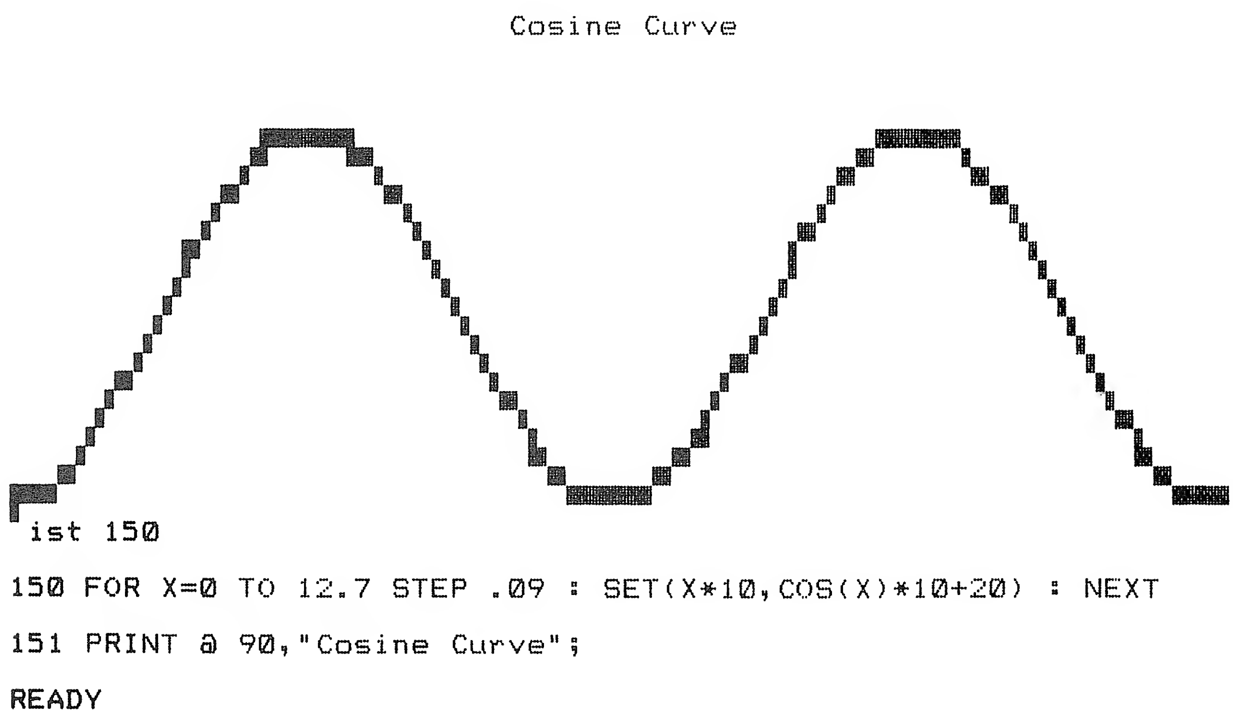


Figure 11-9. Model I, III, 4 Standard Graphics Screen Print Example

LOW-RESOLUTION GRAPHICS

Low-resolution graphics in the MC-10 and Color Computer use the text screen of either computer. The *text screen* is the screen used to display BASIC text and for system messages and commands. It's made up of 16 lines with each line containing 32 character positions, as shown in Figure 11-10. A single text character or graphics character can be held in each character position.

Displaying text on the screen can be done by a PRINT statement. The text will scroll as more lines are printed on the screen. A PRINT @ command can be used to display text at any of the 512 character positions on the screen. PRINT @128, "TEXT", for example, will print TEXT at location 128, the start of the fifth line. Text can also be printed by doing a POKE command directly to one of the screen memory locations. The codes used for the text screen, however, are somewhat different from normal ASCII representation, as shown in Table 11-1.

The Color Computer currently doesn't have a lowercase display. However, lowercase can be entered for printing by using inverse video characters, as shown in the table. Inverse video characters are green on black, instead of black on green and are selected by holding down the SHIFT key, followed by the 0 key. Pressing the two keys again will restore the normal video.

The text screen is a part of RAM memory and characters can also be stored into the text screen by a POKE command. The characters POKEd will correspond to the codes given in Table 11-1. The text screen is at location 1024, as shown in Figure 11-11.

Printer Hint

HOW THE MODEL I, III, 4 STANDARD GRAPHICS SCREEN PRINT PROGRAM WORKS

This program is similar to the block graphics screen print program. In place of a graphics block, however, a seven-dot column graphics pattern is printed.

The program scans the video memory at RAM locations 15360 through 16383 by a PEEK from each location. The video memory is scanned a row at a time. For each row, three row segments are constructed and held in ZZ$(1) through ZZ$(3), a three-element string array. If the current character is displayable, it is added to the ZZ$(2) string and blanks are added to ZZ$(1) and ZZ$(3). If the current character is a graphics character, the proper string for the graphics pattern is added to ZZ$(1), ZZ$(2), and ZZ$(3). At the end of each row, ZZ$(1), ZZ$(2), and ZZ$(3) are printed out in three graphics-mode line feeds. As each ZZ$ string is printed, it is set to a "null" value in preparation for the next line. Graphics mode is switched on and off, depending upon whether a graphics pattern or printable character is being printed.

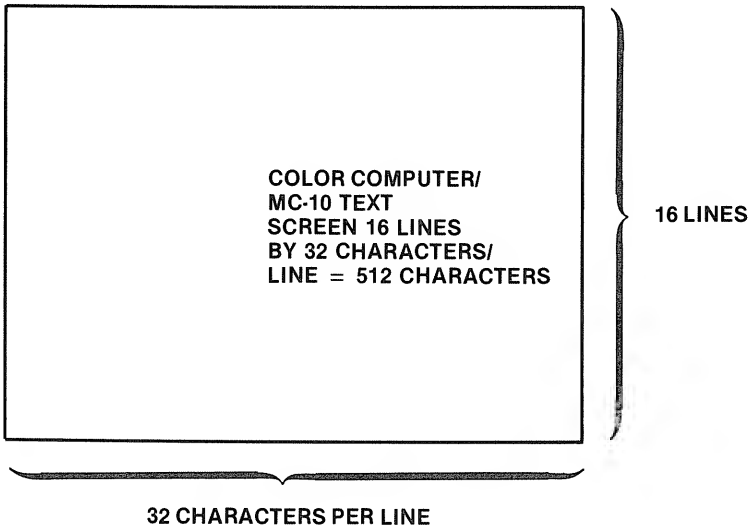


Figure 11-10. Color Computer/MC-10 Text Screen

Table 11-1. Color Computer/MC-10 screen codes

Green on Black (Inverted)				Black on Green (Normal)			
0	@	32	(blank)	64	@	96	(blank)
1	A	33	!	65	A	97	!
2	B	34	"	66	B	98	"
3	C	35	#	67	C	99	#
4	D	36	$	68	D	100	$
5	E	37	%	69	E	101	%
6	F	38	&	70	F	102	&
7	G	39	'	71	G	103	'
8	H	40	(	72	H	104	(
9	I	41	)	73	I	105	)
10	J	42	*	74	J	106	*
11	K	43	+	75	K	107	+
12	L	44	,	76	L	108	,
13	M	45	—	77	M	109	—
14	N	46	.	78	N	110	.
15	O	47	/	79	O	111	/
16	P	48	0	80	P	112	0
17	Q	49	1	81	Q	113	1
18	R	50	2	82	R	114	2
19	S	51	3	83	S	115	3
20	T	52	4	84	T	116	4
21	U	53	5	85	U	117	5
22	V	54	6	86	V	118	6
23	W	55	7	87	W	119	7
24	X	56	8	88	X	120	8
25	Y	57	9	89	Y	121	9
26	Z	58	:	90	Z	122	:
27	[	59	;	91	[	123	;
28	\	60	<	92	\	124	<
29	]	61	>	93	]	125	>
30	↑	62	=	94	↑	126	=
31	←	63	?	95	←	127	?

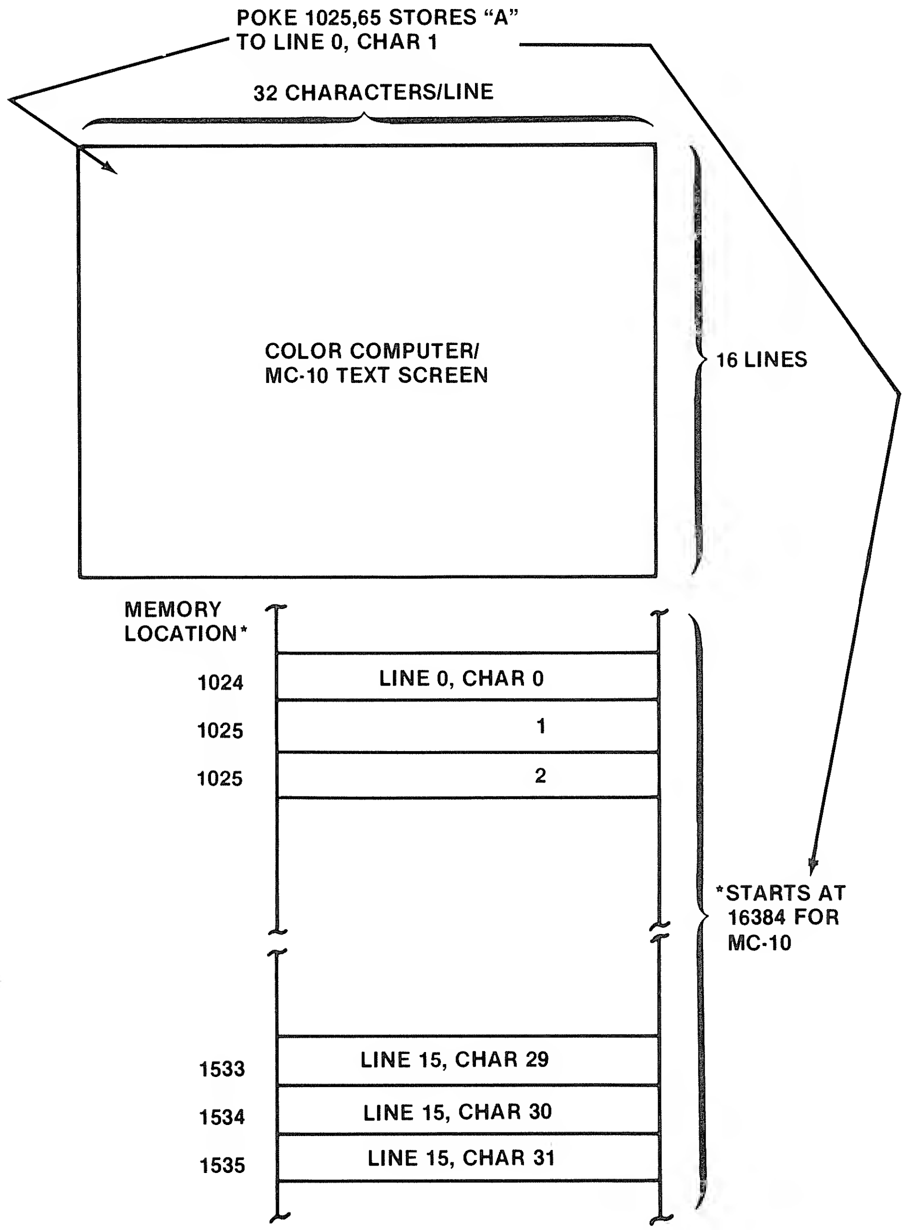


Figure 11-11. POKEing to the Color Computer Text Screen

Each of the 512 character positions can also be used to hold a graphics character instead of a text character. Graphics characters are shown in Figure 11-12. There are four graphics elements per character, in two rows by two columns. A graphics character can be one of eight colors; the color for the character is selected by coding the proper color code into the character.

A graphics element can be set by using the SET command. The SET command sets a specific graphics element in a column, row position to a specified color. SET 63,32,3 for example, sets the last graphics element on the screen to blue. There are 64 columns for graphics elements because there are two elements per horizontal character position. There are 32 rows

Printer Hint

COLOR COMPUTER GRAPHICS

The Color Computer has a number of different graphics modes. *Low-resolution* graphics works with the text screen and allows only 64-by-32 elements. Each character position must have the same color, but there are eight colors allowed. Text may be mixed in with the graphics in low-resolution mode.

High-resolution graphics works with graphics screens that are much larger than the text screen area. In this mode, two or four colors are allowed, but no text. The advantage of high-resolution graphics is that it's very dense—256 by 192 pixels in two colors. There are also a variety of BASIC commands to draw circles, ellipses, lines, and rectangles and to paint them with colors.

For a complete discussion of Color Computer graphics, see my Radio Shack book titled (believe it or not) "Color Computer Graphics."

POSSIBLE GRAPHICS CHARACTERS FOR ANY CHARACTER POSITION

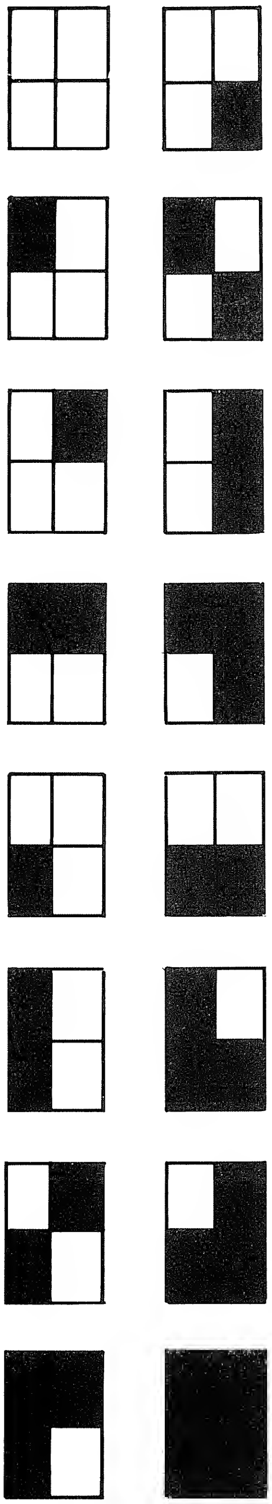


Figure 11-12. Color Computer/MC-10 Text Screen Graphics Characters

for graphics elements, because there are two elements per vertical character position.

Graphics elements can also be set by using POKEs. The format for the POKE is POKE loc,value, where "loc" is a memory location in the text screen from 1024 through 1535, and "value" is a coded value for the pixels to be set and the color. The code for the POKE is determined by adding 128, plus a color code, times eight plus a weight value for the graphics element to be set. The weights to be used are shown in Figure 11-13. The weight values of 1, 2, 4, and 8 are added to the 128 value and the color code, and the result is POKEd into the text screen location.

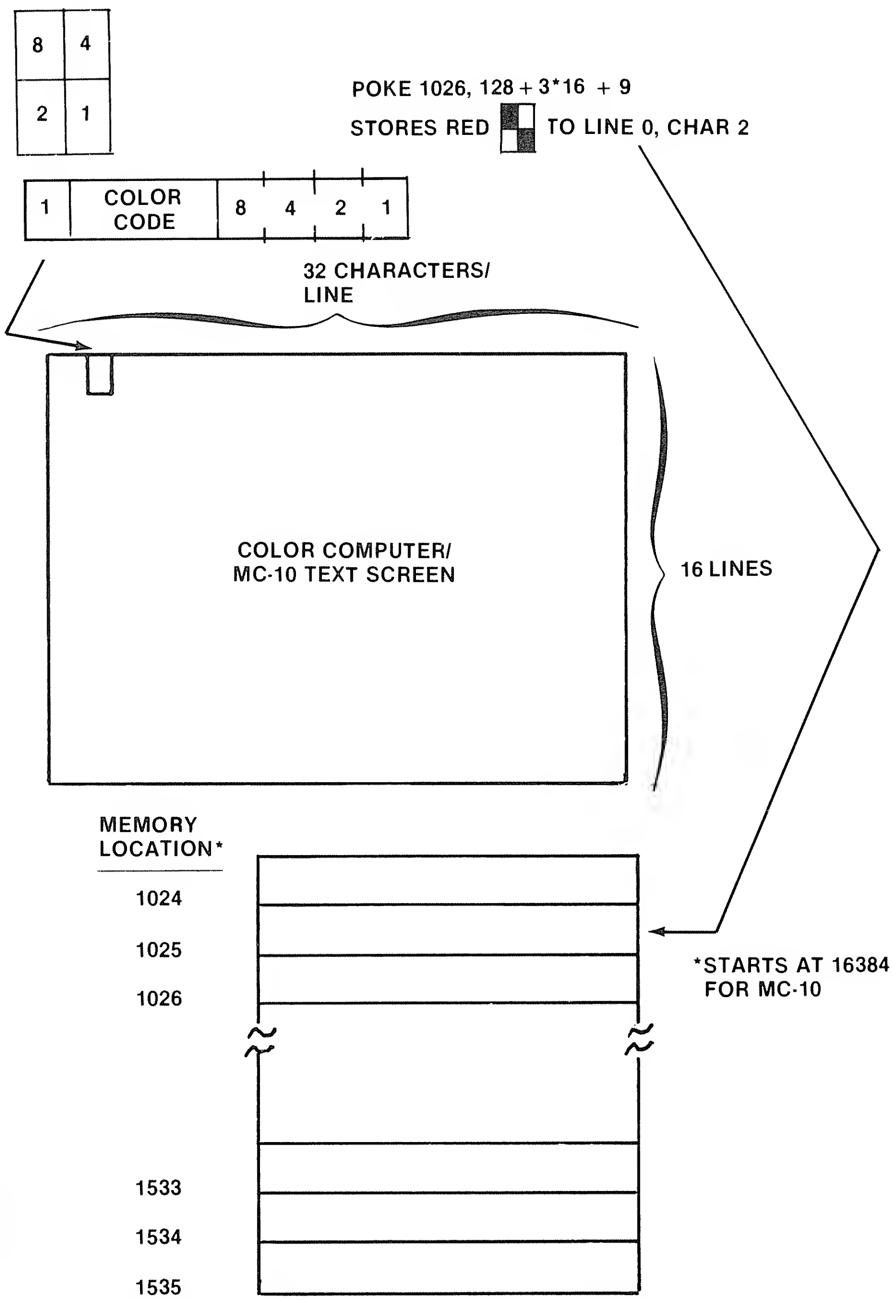


Figure 11-13. Color Computer/MC-10 Graphics Character Weights

Another method for setting graphics elements uses the CHR$ function. The value for the CHR$ is calculated, as in the POKE case, and used in a string. The string "THIS IS TEXT "+CHR$(134)+" AND GRAPHICS" results in the intermixed text and graphics shown in Figure 11-14.

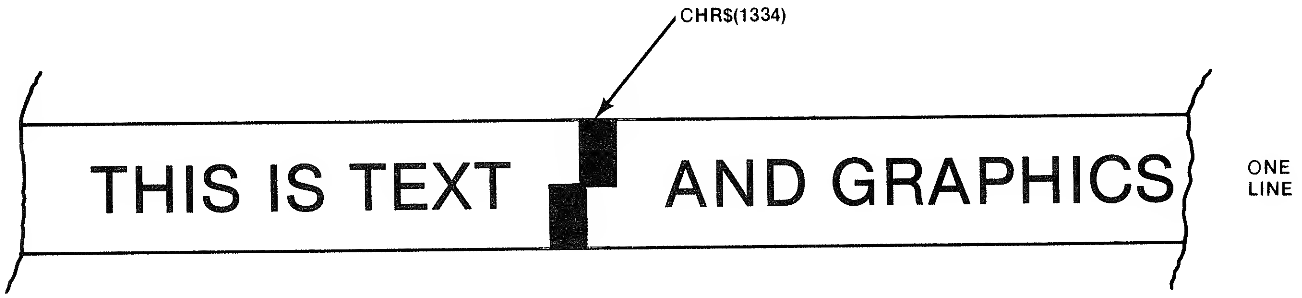


Figure 11-14. Color Computer/MC-10 CHR$ Use for Graphics

GENERAL SCHEME FOR PRINTING GRAPHICS

The general approach to printing text screen graphics from the screen is to read directly all of the 1024 screen locations from video memory at 1024 through 1535 by using PEEKs. The PEEK command reads the contents of a memory location. The program can then decode the graphics value from the PEEK variable and print out a corresponding block graphics or high-resolution graphics character to reproduce the screen.

One of the biggest problems in dumping the screen is achieving the same proportions on the printout as appear on the screen. The Color Computer and MC-10 screen have an aspect ratio of about 4-to-3—4 horizontal units to 3 vertical units. This makes a graphics element about 4/64, or 2/32 of a unit horizontally by 3/32 of a unit vertically, roughly a 2-to-3 aspect ratio for each graphics element. To reproduce the screen on a printer, the program must try to get the same proportions on the printout of graphics elements.

USING BLOCK GRAPHICS TO PRINT OUT A TEXT SCREEN

Block graphics are discussed in "Using Block Graphics," in this section. You might want to read that material to get a background on the block graphics characters. Block graphics are available in the newer printers and are really a subset of characters in the range 224 to 254. They are printed not in graphics mode, but in normal text mode.

The elements in the block graphics characters print at about 1/20 of an inch horizontally by 1/24 of an inch vertically. That's an aspect ratio of (1/20):(1/24) or about 24/20, or 6 to 5. However, if two vertical block elements are used for every screen element, the ratio becomes 3 to 5. Although this is not equal to the aspect ratio of the screen, the 3-to-5 ratio is acceptably close to the 2-to-3 screen ratio.

The program in Listing 11-3 uses block graphics characters to print a screen of graphics characters or text characters. The program is a subroutine that you can call from your own BASIC program, or which can stand alone. (For the program to stand alone, delete the RETURN line.) Figure 11-15 shows an actual printout of the Color Computer screen from this program. If you will be using the program on the MC-10, change the "1024" in line 10040 to "32768".

Printer Hint

HOW THE COLOR COMPUTER/MC-10 BLOCK GRAPHICS SCREEN PRINT PROGRAM WORKS

This program scans the video memory, at RAM locations 1024 through 1535, by a PEEK from each location. The video memory is scanned a row at a time. For each row, two row segments are constructed and held in ZZ$(1) and ZZ$(2), a two-element string array. If the current character is a displayable character, it is added to the ZZ$(2) string and a blank is added to ZZ$(1). If the current character is a graphics character, the proper block graphics character is added to ZZ$(1) and ZZ$(2). At the end of each row, ZZ$(1), and ZZ$(2) are printed out in two half-line feed lines. As each ZZ$ string is printed, it is set to a "null" value in preparation for the next line.

```
10000 'SUB COLOR COMPUTER BLOCK GRAPHICS SCREEN PRINT
10010 CH$(0)=CHR$(224) : CH$(1)=CHR$(234) : CH$(2)=CHR$(233) : CH$(3)=CHR$(239)
10015 PRINT #-2,CHR$(27);CHR$(28); 'SET HALF LINE FEED
10020 FOR ZY=0 TO 15
10030 FOR ZX=0 TO 31
10040 ZP=1024+ZX+ZY*32
10041 IF PEEK(ZP)>127 THEN 10050
10042 ZP=PEEK(ZP)
10043 IF ZP<32 THEN ZP=ZP+96 : GOTO 10045
10044 IF ZP>95 THEN ZP=ZP-64
10045 ZZ$(1)=ZZ$(1)+" " : ZZ$(2)=ZZ$(2)+CHR$(ZP) : GOTO 10070
10050 ZZ$(1)=ZZ$(1)+CH$((PEEK(ZP) AND 15)/4) : ZZ$(2)=ZZ$(2)+CH$(PEEK(ZP) AND 3)
10070 NEXT ZX
10075 FOR ZX=1 TO 2 : PRINT #-2,ZZ$(ZX) : ZZ$(ZX)="" : NEXT
10080 NEXT ZY
10090 PRINT #-2,CHR$(27);CHR$(54); 'SET FULL LINE FEED
10100 RETURN
```

Listing 11-3. Color Computer/MC-10 Block Graphics Screen Dump Program

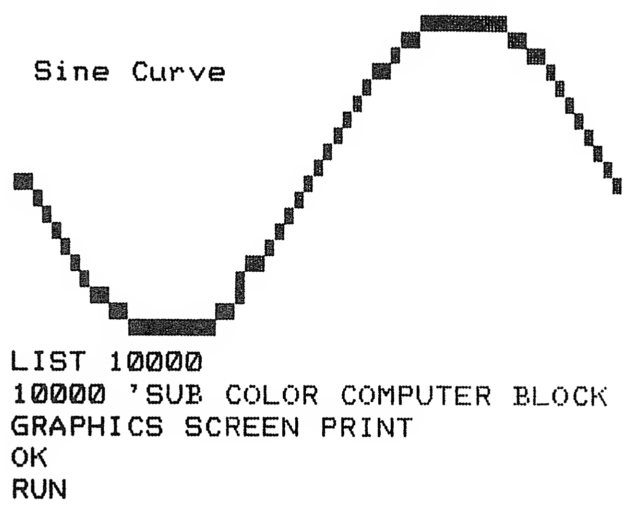


Figure 11-15. Color Computer/MC-10 Block Graphics Screen Print Example

HIGH-RESOLUTION GRAPHICS IN THE COLOR COMPUTER

The second type of graphics available in the Color Computer is high-resolution graphics. High-resolution graphics includes graphics modes that range from a 64-by-64 screen resolution in four colors, to a 256-by-192 one-color resolution. When these modes are in force (set by the PMODE command in BASIC), there are two types of screen areas in memory—the text page and one or more graphics pages. The BASIC program can switch back and forth between the text screen and graphics screen at will via the SCREEN command. The text screen is used for text entry and display, and the graphics screens are used for display of pure graphics data without text (unless the text is generated by the graphics program).

Graphics modes are either four-color or two-color. When four colors are used, the graphics page is divided up into elements of two bits each, or four elements per byte within the page, as shown in Figure 11-16. Each element

of two bits holds the color code for the screen element. When a two-color mode is used, there are eight elements per byte, or one bit per element, as shown in the figure.

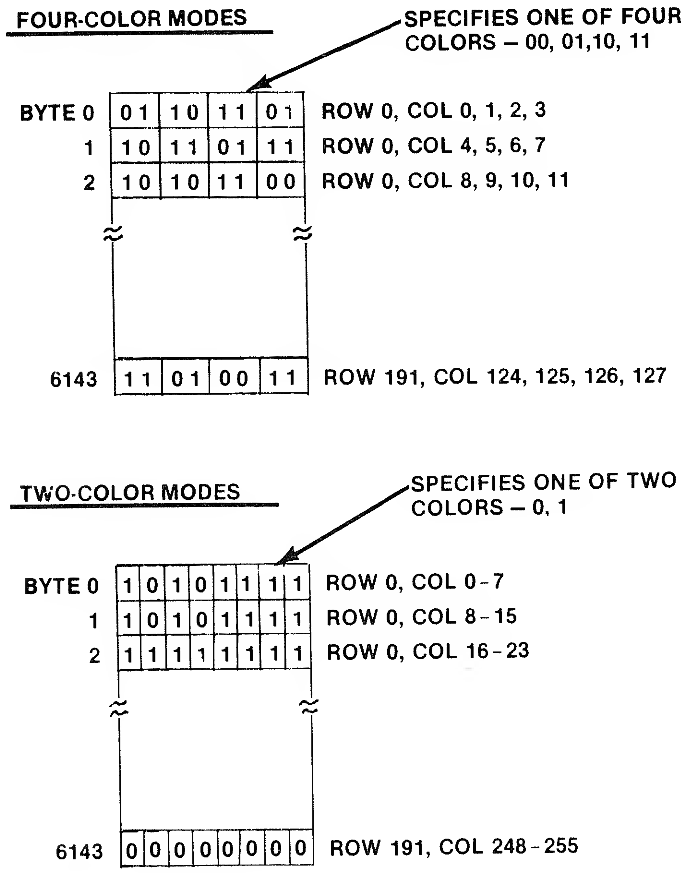


Figure 11-16. Color Computer High-Resolution Graphics

The first row of the graphics screen, in either a two-color or four-color mode, is found in the first 64, 128, or 256 elements in the graphics page, depending upon the number of pixels in the graphics resolution selected. In a 128-by-192 four-color mode, for example, the first row is in the first 128-by-2 bits (= 256 bits = 32 bytes) of the graphics screen. The next row is in the next 32 bytes, and so forth.

Printing a graphics screen involves scanning the graphics screen area in memory, element by element, finding the graphics color code for each element, and then using block graphics or high-resolution graphics to print out a simulation of the graphics pixel, or picture element.

As in the case of a text screen, one of the biggest problems in such a print screen program is matching the proportions of the screen. Another problem is that, except for the CGP-220 Ink-Jet Printer, the colors on the

screen can't be reproduced. We'll consider a two-color 256-by-192 screen print program here to avoid the problem of what to do about colors, and assume that you'll probably want to use the program to print the highest-resolution screen.

A 256-by-192 screen print program will print 256 graphics pixels for every row. We could use block graphics for this, but since each element of a block graphics character is 1/20 of an inch wide at 10 pitch, the resulting printed row would be 12.8 inches wide, too large for most printers. A better solution is to use high-resolution print graphics where the typical number of columns per horizontal inch is 100. High-resolution print graphics uses a half line feed of about 0.1 to 0.12 inch, making the number of dots per vertical inch 70 to 83. Printing two vertical dots for every screen pixel results in 192 times 2, or 384 dots total. When the screen is printed out, the height of the picture will be 384/83 to 384/70, or about 4.6 to 5.5 inches.

What about the horizontal dimension, which should be about 6.1 to 7.3 inches, to preserve the screen proportions of 4-to-3? Printing two columns for every horizontal pixel would give us 512 columns per picture about 8.5 inches at 60 columns per inch, somewhat wider than optimum, but adequate.

The program shown in Listing 11-4 prints a 256-by-192 two-color screen on your Color Computer printer. It takes a long time to process the 49,152 (!) points on the screen, so be prepared to wait for a while before getting the finished printout. The program is set up as a subroutine which you can call from your own BASIC code, or you can delete the RETURN line for it to stand alone. Figure 11-17 shows an actual printout from the program.

```
10000 'HI RES SCREEN PRINT
10010 ZM=1536 : PRINT #-2,CHR$(18); *
10020 FOR ZY=0 TO 191 STEP 3.5: FOR ZX=0 TO 31 : ZZ=ZM+ZX+INT(ZY)*32
10030 ZC(1)=PEEK(ZZ) : ZC(2)=PEEK(ZZ+32) : ZC(3)=PEEK(ZZ+64) : ZC(4)=PEEK(ZZ+96)
10040 FOR ZW=7 TO 0 STEP -1 : ZZ=2^ZW
10050 IF (ZC(1) AND ZZ)>0 THEN ZD(1)=1 ELSE ZD(1)=0
10060 IF (ZC(2) AND ZZ)>0 THEN ZD(2)=1 ELSE ZD(2)=0
10070 IF (ZC(3) AND ZZ)>0 THEN ZD(3)=1 ELSE ZD(3)=0
10080 IF (ZC(4) AND ZZ)>0 THEN ZD(4)=1 ELSE ZD(4)=0
10090 IF ZY  =INT(ZY  ) THEN ZD$=CHR$(128+ZD(1)*3+ZD(2)*12+ZD(3)*48+ZD(4)*64)
10100 IF ZY  <>INT(ZY  ) THEN ZD$=CHR$(128+ZD(1)*1+ZD(2)*6+ZD(3)*24+ZD(4)*96)
10110 PRINT #-2,ZD$;ZD$; : NEXT ZW,ZX : PRINT #-2 : NEXT ZY
10120 PRINT#-2,CHR$(30); : RETURN
```

* 'ZM = FOR DISK BASIC

Listing 11-4. Color Computer High-Resolution Screen Dump Program

TO PRINT THE SCREEN IN THE MC-10 AND COLOR COMPUTER

The TP-10 can be used as a normal printer to print text with 32 characters per line (16 in the elongated mode) and with a standard line spacing of six lines per inch. However, unlike other Radio Shack printers, it will print an almost exact copy of an MC-10 or Color Computer screen, complete with low-resolution graphics. (Of course, on the MC-10 only low-resolution graphics is supported.)

The 6-inch line spacing on the TP-10 means that 16 lines can be printed in an area 2.66 inches high. Since the character spacing is 10 pitch (ten

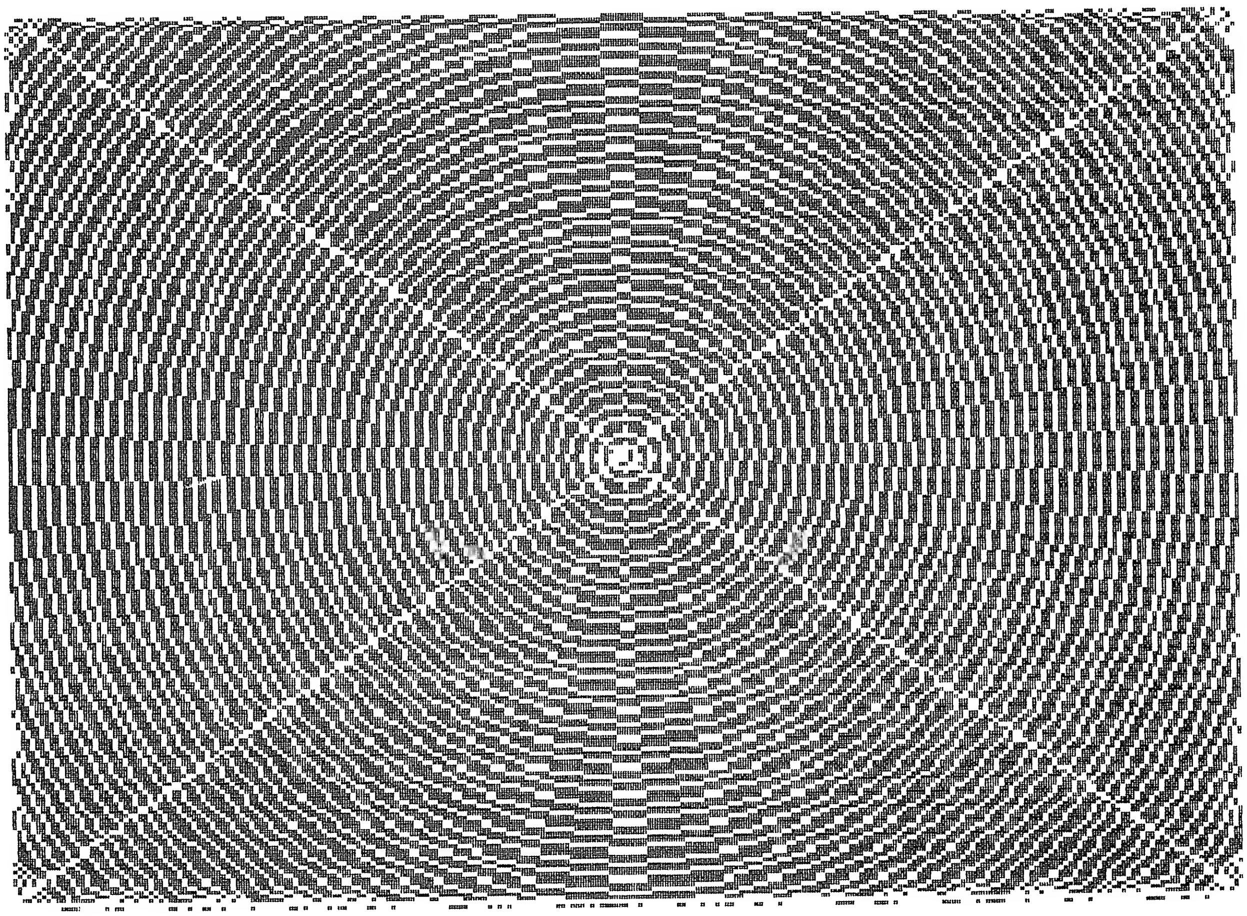


Figure 11-17. Color Computer High-Resolution Screen Print

characters per inch), the 32 characters per line of the MC-10 or Color Computer can be printed in an area 3.2 inches wide. The aspect ratio or proportion of width to height, is 4-to-3.3, very close to the screen aspect ratio of 4-to-3. The printout closely resembles the screen.

The character set of the TP-10 is shown in Table 11-2. It isn't too exciting until you reach the upper 128 characters within the range of 128 through 255. These characters are block graphics characters which reproduce the two-by-two graphics-elements-per-character position of the MC-10 or the low-resolution graphics mode of the Color Computer. The block graphics characters use three-by-six dots for the left-hand blocks and four-by-six dots for the right-hand blocks—not exact proportions, but close enough. See Figure 11-18.

There are 16 different configurations of a two-by-two graphics matrix, and the character set includes all 16. In the MC-10, or the low-resolution mode of the Color Computer, graphics characters are displayed by PRINTing a CHR$() value on the screen. The value is formed as shown in Figure 11-19.

The graphics elements in each character position are assigned weights of 1, 2, 4, and 8. To find the value for CHR$(), add together the weights, and

Table 11-2. TP-10 Character Set

Code Dec.	Code Hex.	Char.	Code Dec.	Code Hex.	Char.	Code Dec.	Code Hex.	Char.
32	20	(Space)	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	F	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	—	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	T	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	,	91	5B	[	123	7B	{
60	3C	<	92	5C	\	124	7C	!
61	3D	=	93	5D	]	125	7D	}
62	3E	>	94	5E	^	126	7E	∿
63	3F	?	95	5F	—			

Dec.					Hex.			Char.
128 192	144 208	160 224	176 240	80 C0	90 D0	A0 E0	B0 F0	
129 193	145 209	161 225	177 241	81 C1	91 D1	A1 E1	B1 F1	
130 194	146 210	162 226	178 242	82 C2	92 D2	A2 E2	B2 F2	
131 195	147 211	163 227	179 243	83 C3	93 D3	A3 E3	B3 F3	
132 196	148 212	164 228	180 244	84 C4	94 D4	A4 E4	B4 F4	
133 197	149 213	165 229	181 245	85 C5	95 D5	A5 E5	B5 F5	
134 198	150 214	166 230	182 246	86 C6	96 D6	A6 E6	B6 F6	
135 199	151 215	167 231	183 247	87 C7	97 D7	A7 E7	B7 F7	
136 200	152 216	168 232	184 248	88 C8	98 D8	A8 E8	B8 F8	
137 201	153 217	169 233	185 249	89 C9	99 D9	A9 E9	B9 F9	
138 202	154 218	170 234	186 250	8A CA	9A DA	AA EA	BA FA	
139 203	155 219	171 235	187 251	8B CB	9B DB	AB EB	BB FB	
140 204	156 220	172 236	188 252	8C CC	9C DC	AC EC	BC FC	
141 205	157 221	173 237	189 253	8D CD	9D DD	AD ED	BD FD	
142 206	158 222	174 238	190 254	8E CE	9E DE	AE EE	BE FE	
143 207	159 223	175 239	191 255	8F CF	9F DF	AF EF	BF FF	

then add 128 to mark the character as graphics. You'll also have to add a color code of 0 to 112, as shown in the figure. That's how the *screen* value is calculated. The CHR$() for the TP-10 *printer* is calculated much the same way, but without the Color Code. This means that the values for the 16 graphics blocks are 128 (no graphics elements on) to 143 (all elements on). Since the color code is ignored in the printer codes, however, the next 16 codes from 144 through 159 *also* print the same graphics characters, and the next 16 from 160 through 175, and so forth. This means that you can go directly to the MC-10 or Color Computer screen, find the CHR$() value at each character position, and print it directly without having to worry about the color code. Whatever the color code is on the screen, the printer will reproduce every graphics element that is on and will not print off elements. (Background colors will not print.)

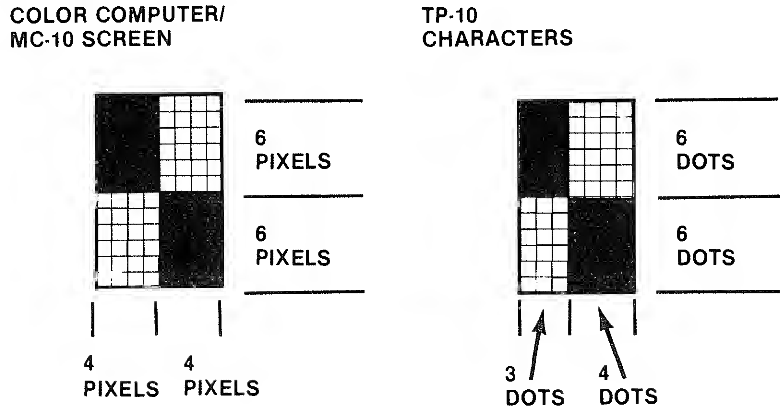


Figure 11-18. TP-10 Block Graphics Characters

Here's a simple program to print the MC-10 screen or the low-resolution screen of the Color Computer. It will print both text and low-resolution graphics.

```
10000 ' MC-10/COLOR COMPUTER SCREEN PRINT FOR THE TP-10
10010 FOR ZL=0 TO 15
10020 FOR ZC=0 TO 31
10030 ZA=PEEK(1024+(ZL*32)+ZC)
10040 IF ZA>127 THEN 10070
10050 IF ZA<32 THEN ZA=ZA+96
10060 IF ZA>95 THEN ZA=ZA-64
10070 PRINT#-2,CHR$(ZA);
10080 NEXT ZC
10090 PRINT#-2
10100 NEXT ZL
10110 RETURN
```

Change the "1024" in line 10030 for an MC-10 system. The program is designed to be called from your main program as a subroutine. Delete line 10100 to use the program "in line" with your other BASIC code.

The program PEEKs the values from the screen RAM area, then sends the values directly to the printer via a CHR$(). An actual print of the screen by this program is shown in Figure 11-20. Compare this simple program with some of the other programs for printing the screen in "Printing Graphics Screens." This one is much less complicated and gives almost true proportion.

Printer Hint

HOW THE COLOR COMPUTER HIGH-RESOLUTION SCREEN PRINT PROGRAM WORKS

The program processes three graphics rows at a time. Each of the 192 graphics rows on a graphics screen is made up of 32 bytes of 8-bits each, for a total of 256 bits across, in the highest-resolution mode. For each of 64 iterations in the program for three rows, an inner *loop* of 32 iterations processes each of the 32 bytes per row, and a third loop processes bits horizontally.

In the innermost loop, the four values in the ZC array are set equal to the bit status of the vertical column of *four* bits, starting from the current row position. Although only three column bits are completely processed, either the first or last bit of the column must be printed in a "half pattern." Each PRINT#-2, prints a vertical graphics pattern representing 3½ column bits. This is repeated for the eight bits-per-row byte, for the 32 bytes per row, and for the 64 sets of three rows each.

Printer Hint

COLOR COMPUTER/MC-10 SCREEN CHARACTERS

The Color Computer/MC-10 screen characters are quite different from normal ASCII and that's why we went through some gyrations in the TP-10 screen print program. The basic premise is that both black on green, the normal text display, and green on black characters can be represented in codes from 0 through 127. This requires some fancy footwork in converting the actual screen values to ASCII for printer use, but it's not too involved.

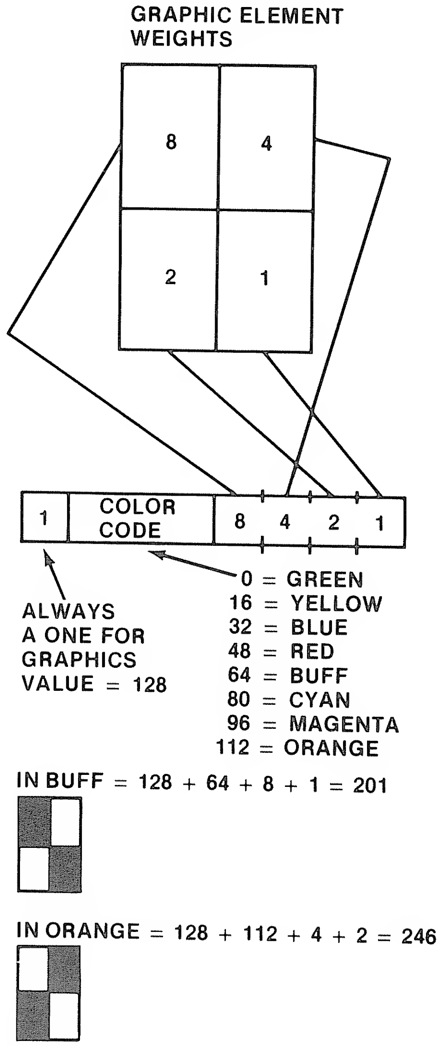


Figure 11-19. Color Computer/MC-10 Low-Resolution Graphics Weights

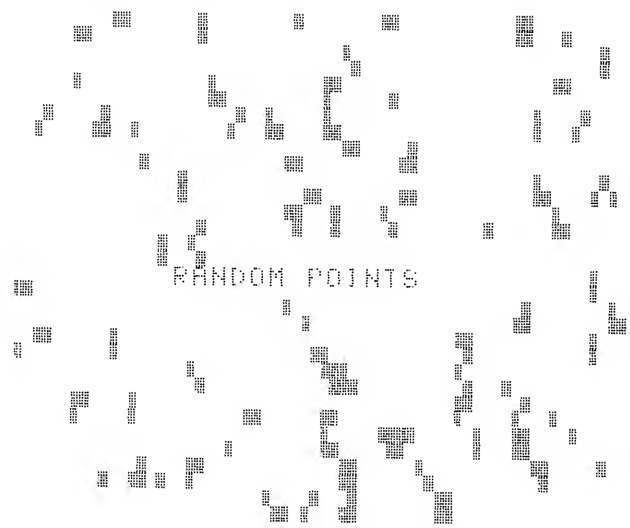


Figure 11-20. TP-10 Screen Print for Color Computer/MC-10

CHAPTER 12

CREATIVE GRAPHICS

You can use the graphics mode to draw vertical lines, horizontal lines, and to create forms and pictures. Before we describe how to do this in graphics mode, first consider whether using block graphics might be a better way to do this. Block graphics are discussed in "Using Block Graphics," and are available on most of the newer printers. If your printer doesn't have block graphics, however, like the DMP-100 (DMP-VII), then you can still produce nice forms using graphics alone.

If you don't understand how graphics work, read "Using the Graphics Modes" to get some background on the basics of graphics. If you have a daisy-wheel printer, see "Plotting Along with Dot-Matrix and Daisy-wheel Printers" for a method to draw lines with daisy-wheels.

DRAWING HORIZONTAL LINES USING GRAPHICS

This is the easiest instance in using graphics, since a one dot pattern can be repeated across the paper to produce the line, as shown in Figure 12-1.

To draw the line, you must first decide how far down on the paper you want the line to be drawn, as shown in the figure. Because you'll be using graphics lines, which are not a full line feed, you'll have to space the line down by a series of carriage returns, after setting graphics mode by CHR$(18). The line-feed vertical spacing increments range from 0.1 to 0.12 inches depending upon your printer. Next, you'll have to decide how far in from the left edge of the paper you want the line to start. You can then space to the dot column position by a position function, with the proper dot column specified. The position is done by a CHR$(27);CHR$(16);CHR$(MM);CHR$(LL), where MM and LL are found by dividing the column position you want by 256, and using the quotient as MM and the remainder as LL. To space to the 402 dot column, for example, divide 402 by 256 to get an MM

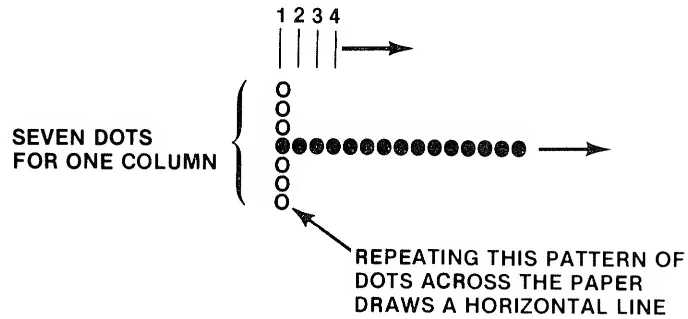


Figure 12-1. Drawing Horizontal Lines in Graphics Mode

of 1 and an LL of 146. The dot column spacing will vary with your printer type and the pitch which you are using. See Table 10-2 to find the proper column position for your printer.

Note: Because the positioning escape sequence contains numeric values that may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, identical to the top-of-form character, the printer driver will attempt to execute a page-eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line-feed character, it will be changed to a 13 in the printer driver. Avoid escape sequences containing these values by using a slightly larger or smaller value in the CHR$() numeric variable, or use a combination of two or more escape sequences.

The next step is to decide which dot or dots you want for the line, as shown in Figure 12-2. The spacing of the seven dots in a graphics column is about 0.1 inch, so if you use one dot the line will be about 1/70 of an inch high. You may want to use more than one dot, to get a thicker line. The dots are weighted according to their position from top to bottom. The top dot is a weight of 1, the next is a weight of 2, the next is 4, the next is 8, the next is a 16, the next is 32, and the seventh (bottommost) dot is a weight of 64. Add together the weights for the dots you want printed, and then add a value of 128 to the result, to indicate that the character is a graphics character.

If you wanted the top and bottom lines printed, for example, you'd have 1 + 64 + 128, or CHR$(193).

Next, decide how long you want the line to be, in dot columns. Again, this depends upon your printer type and column spacing. After you've figured out the line length in dot columns, you can use the repeat function to repeat the CHR$() value in groups of 255. The repeat function uses the code sequence CHR$(28);CHR$(NN);CHR$(CC), where NN is the number of times for the repeat 1-255, and CC is the value of the character to be repeated. In this case the CC value will be the value you calculated for the dots you wanted printed.

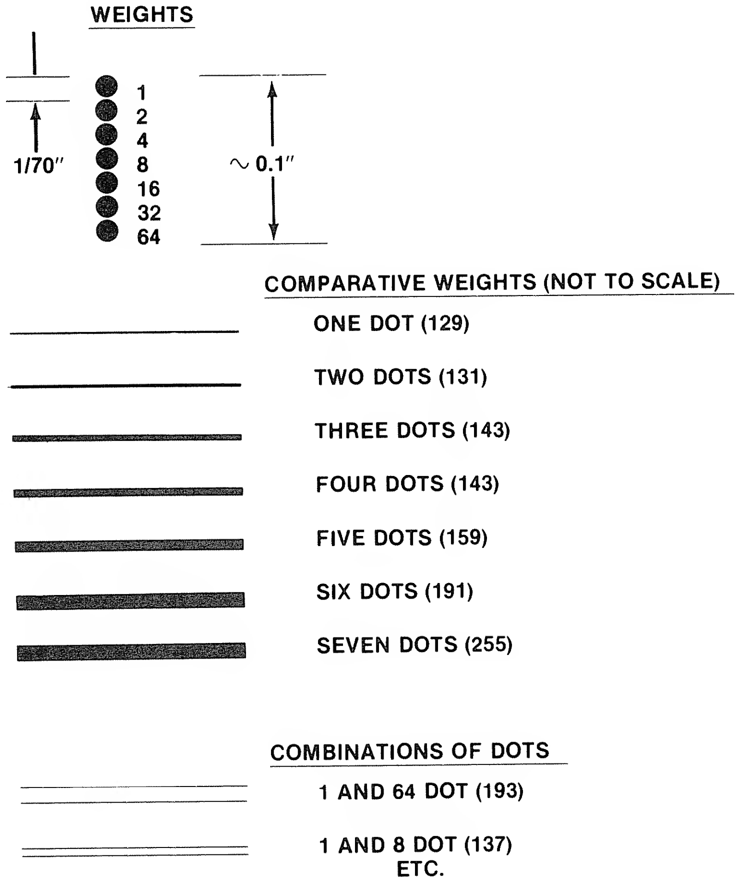


Figure 12-2. Using Dots for Horizontal Lines

Note: Because the repeat sequence contains numeric values that may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, identical to the top-of-form character, the print driver will attempt to execute a page eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line-feed character, it will be changed to a 13 in the print driver. Avoid escape sequences containing these values by using a slightly larger or smaller value in the CHR$() numeric variable, or use a combination of two or more repeats to get the same result. Another alternative is to use the BASIC STRING$ command to repeat from within BASIC.

Using the double line as an example, the following short program will print across the page, starting at 16 graphics lines down (about 1.6 inches on most printers) and column position 102, for 200 dot columns (see Figure 12-3).

```
100 LPRINT CHR$(18)
110 FOR I=1 TO 16
```

```
120 LPRINT
130 NEXT I
140 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(102);
150 LPRINT CHR$(28);CHR$(200);CHR$(193)
```

PRINTING VERTICAL LINES

Printing vertical lines is just about as easy as printing horizontal lines. The only difference is that between each print, you have to space to the next graphics line, as shown in Figure 12-4. This will probably involve using a position function to reposition the print head to the proper print column. Suppose you want to draw a vertical line at dot column 406, starting at graphics line 16 and extending through graphics line 48, as shown in Figure 12-5. Here's how it would be done.

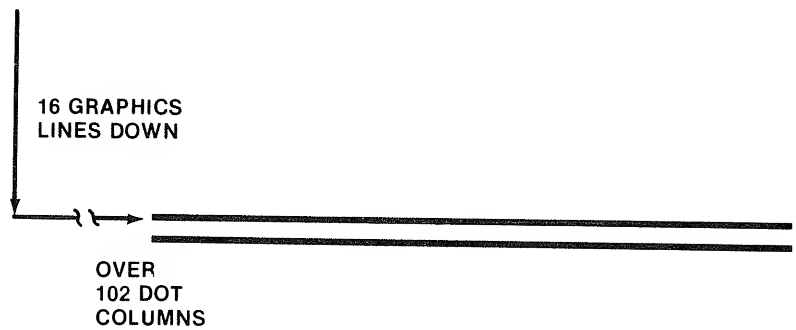


Figure 12-3. Sample Horizontal Lines in Graphics Mode

First, you space down the proper number of lines by a series of 16 line feeds (LPRINTs). Then establish a loop of 33 steps. During each step, position the print head over the proper column by doing a CHR$(27);CHR$(16); CHR$(MM);CHR$(LL), and then fire the print head by doing a CHR$(255), which prints all seven dot columns. For a line that was less than a multiple of seven dots, adjust the top and bottom prints accordingly. We'll use a full column here for simplicity. The final program looks like this:

```
100 LPRINT CHR$(18)
110 FOR I=1 TO 16
120 LPRINT
130 NEXT I
140 FOR I=1 TO 33
150 LPRINT CHR$(27);CHR$(16);CHR$(1);CHR$(146);CHR$(255)
160 NEXT I
```

DRAWING FORMS USING GRAPHICS

If you can't use the forms program in "Using Block Graphics," (this section), then you can use the following approach. We'll duplicate the form shown in Figure 12-6. First, lay out the form on a sheet of paper. Ideally, the layout sheet would have line spacing equal to the graphics line spacing and column

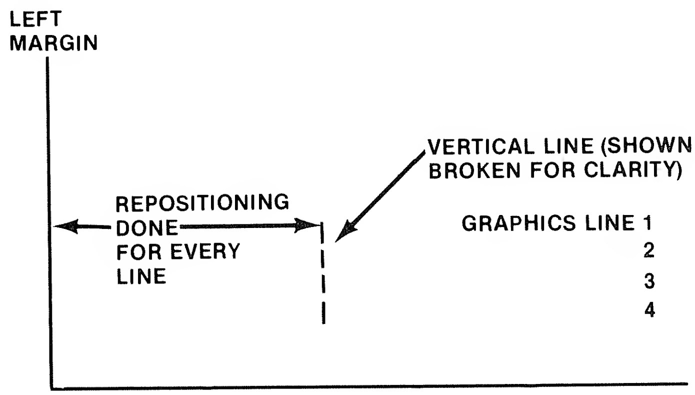


Figure 12-4. Drawing Vertical Lines in Graphics Mode

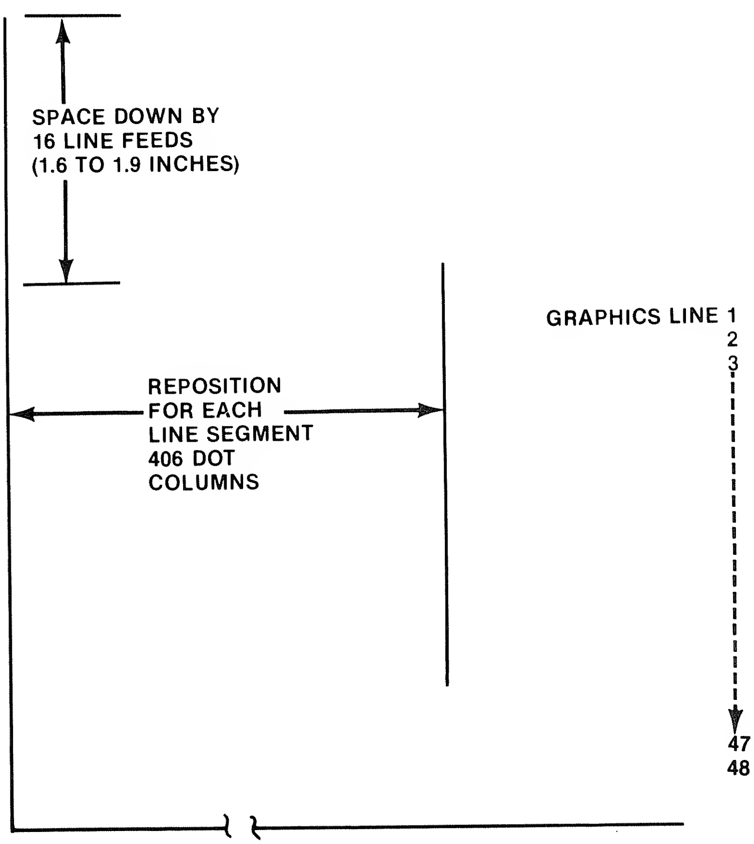


Figure 12-5. Sample Vertical Line in Graphics Mode

numbering to correspond to your printer. If you want text on the form, you'll need to do some planning at this stage. Text can be interspersed with graphics, but you'll have to use three graphics lines to separate the text from the lines, as shown in the figure. Also, you'll have to set data-processing or word-processing mode immediately before the text, and reset it immediately after any text.

From a layout sheet you can determine the spacing for repeat functions to draw the horizontal line segments and to space over to the next vertical line segment. If the form does not start at the left margin, you'll also have to calculate the column positioning for the left edge of the form. Chances are you will be able to use a BASIC subroutine to advantage, also. In the form shown in Figure 12-6, the horizontal lines in the middle of the form all use the same sequence of characters and can be put into a subroutine like this:

```
220 'SUB SIDES AND HORIZONTAL LINES
230 LPRINT CHR$(255);STRING$(200,129);STRING$(98,129);CHR$(255)
240 RETURN
```

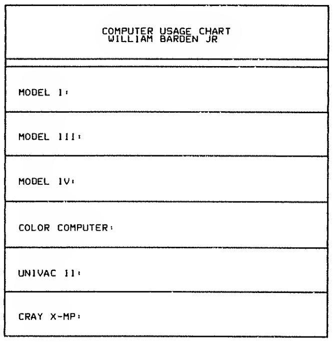


Figure 12-6. Sample Form Done by Graphics

As a matter of fact, the other lines can also be put into a subroutine, except for the line with text in it:

```
250 'SUB SIDES ONLY
260 FOR ZJ=1 TO 2
270 LPRINT CHR$(255);STRING$(200,128);STRING$(98,128);CHR$(255)
280 NEXT ZJ
290 RETURN
```

The top and bottom lines are special cases because of the corners of the form and the intersection of the middle line. The second graphics line also has text in it—a CHR$(30) is done immediately before the text and graphics is then set again after the text.

The final program is shown in Listing 12-1.

PRINTING PICTURES IN GRAPHICS MODE

It's fairly easy to print all kinds of pictures by using the printer graphics mode. Although it's *easy*, we don't want to mislead you—it's also very *tedious* to print pictures, especially if they involve a great deal of nonrepetitive patterns. We'll show you how to produce pictures in the following material.

If you haven't read the material on "Using the Graphics Mode," read it now to get a good understanding of the basic steps in using graphics mode.

If you have a Color Computer, perhaps the easiest way to print pictures is to construct the picture first on your Color Computer screen, and then print the picture by using the screen print program shown in Listing 11-4. However, if you don't own a Color Computer or if you'd rather work directly with printer graphics, read on.

MAKING A PRINTER LAYOUT SHEET

If you're going to print a number of pictures, you will benefit greatly by making up a printer layout sheet. A sample is shown in Figure 12-7. This is an

Printer Hint

PRINTING BAR CODES

It's possible to print *bar codes* with most of the printers currently carried by Radio Shack. One graphics column suffices for the thinnest bar code line, and two- and three-dot columns can be used for the other bars. If you have a Model 100, it might be interesting to experiment with creating your own bar codes and then reading them with the optional bar code reader.

```
100 'COMPUTER USAGE CHART
110 CLEAR 1000
120 LPRINT CHR$(18)
130 GOSUB 220: GOSUB 250
140 FOR ZI=7 TO 8: GOSUB 300: NEXT ZI
150 GOSUB 250: GOSUB 220
160 FOR ZI=1 TO 6
170 GOSUB 220: GOSUB 250: GOSUB 300: GOSUB 250
180 NEXT ZI
190 LPRINT STRING$(200,129);STRING$(100,129)
200 LPRINT CHR$(30)
210 END
220 'SUB SIDES AND HORIZONTAL LINES
230 LPRINT CHR$(255);STRING$(200,129);STRING$(98,129);CHR$(255)
240 RETURN
250 'SUB SIDES ONLY
260 FOR ZJ=1 TO 2
270 LPRINT CHR$(255);STRING$(200,128);STRING$(98,128);CHR$(255)
280 NEXT ZJ
290 RETURN
300 'SUB SIDES AND TEXT
310 IF ZI=1 THEN ZT$="MODEL I:        "
320 IF ZI=2 THEN ZT$="MODEL III:      "
330 IF ZI=3 THEN ZT$="MODEL IV:       "
340 IF ZI=4 THEN ZT$="COLOR COMPUTER:"
350 IF ZI=5 THEN ZT$="UNIVAC II:      "
360 IF ZI=6 THEN ZT$="CRAY X-MP:      "
370 IF ZI=7 THEN ZT$="COMPUTER USAGE CHART"
380 IF ZI=8 THEN ZT$=" WILLIAM BARDEN JR. "
390 IF ZI<7 THEN ZA=12: ZB=196: GOTO 410
400 ZA=89: ZB=89
410 LPRINT CHR$(255);STRING$(ZA,128);
420 LPRINT CHR$(30); 'DISABLE GRAPHICS
430 LPRINT ZT$;
440 LPRINT CHR$(18); 'ENABLE GRAPHICS
450 LPRINT STRING$(ZB,128);CHR$(255)
460 RETURN
```

Listing 12-1. Sample Form Program for Graphics

oversize sheet which has ruled lines representing the column positions and graphics lines in exact proportion to how they will appear on your printer. The sheet shown represents the graphics for a DMP-400. The DMP-400 column density is 100 columns per horizontal inch in condensed mode and the graphics line is 0.1 inch high. The sheet shown in the figure represents a portion of the printed paper, 1.38 inches wide by 1.6 inches high. It has been scaled up so it's actually 7.8 inches wide by 9.3 inches high, or 34 times the area of the actual print area.

The layout sheet makes it easier to convert a figure into the CHR$() values required to produce the graphics picture. As shown in Figure 12-8, the figure to be drawn can be placed on top of the layout sheet, or drawn over the sheet. Once the figure to be drawn is superimposed, the next step is to draw an index, or reference, line to the left of the figure. This reference line is the "0" column position for the columns defining the figure. Next, the CHR$() values are calculated for each line of the figure. (In this case, by the way, the figure is an electronics symbol for a transistor.)

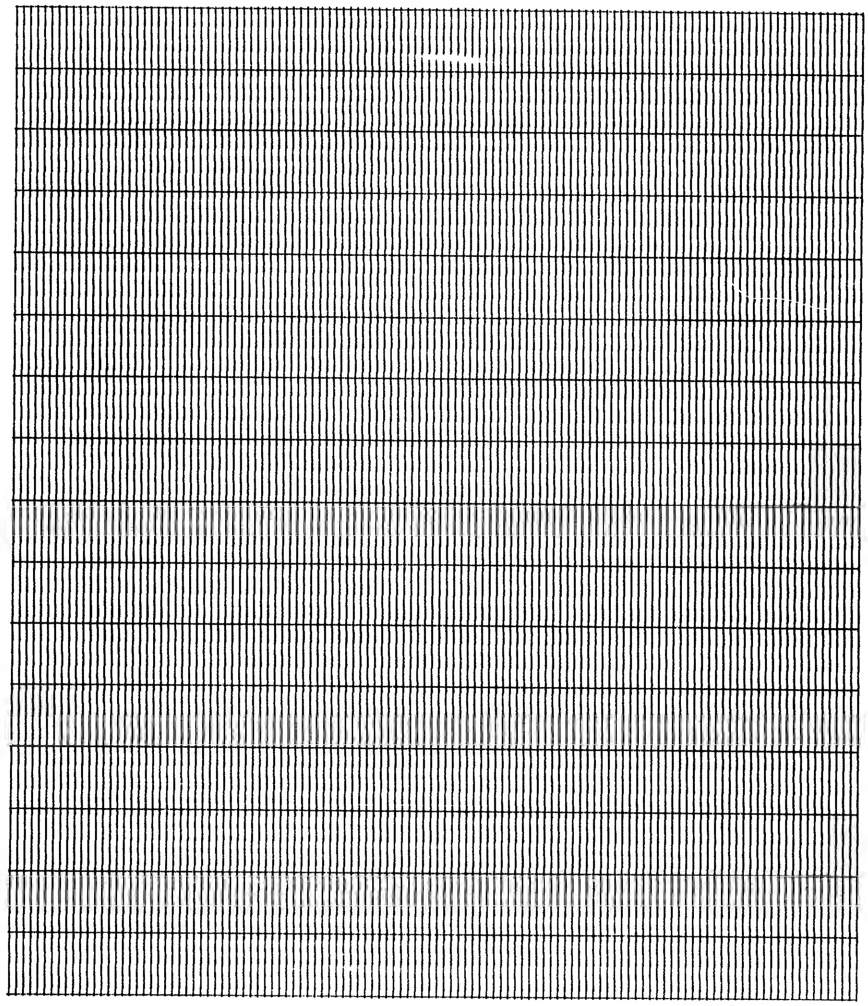


Figure 12-7. Graphics Layout Sheet Sample

CALCULATING CHR$ VALUES

The CHR$ values are calculated by using the weighted dot position method described in "Using the Graphics Mode." In this method, the topmost dot is 1, the next 2, the next 4, the next 8, the next 16, the next 32, and the seventh (bottommost) 64. The final CHR$() value is calculated by adding together all of the dot weights, plus 128 (to indicate graphics mode). Several samples are shown in Figure 12-9.

Next, the horizontally repetitive portions of the figure are found in addition to areas in which a dot column position can be specified. The repetitive portions are patterns that repeat along a graphics line and for which you can use the repeat code of CHR$(28) + CHR$(NN) + CHR$(CC), where NN is the number of times to repeat, and CC is the value to repeat. The repeat code or STRING$ can be used here for blank areas and for several other horizontal lines. In some areas, a column position can be specified to avoid

INDEX LINE = 0 COLUMN POSITION

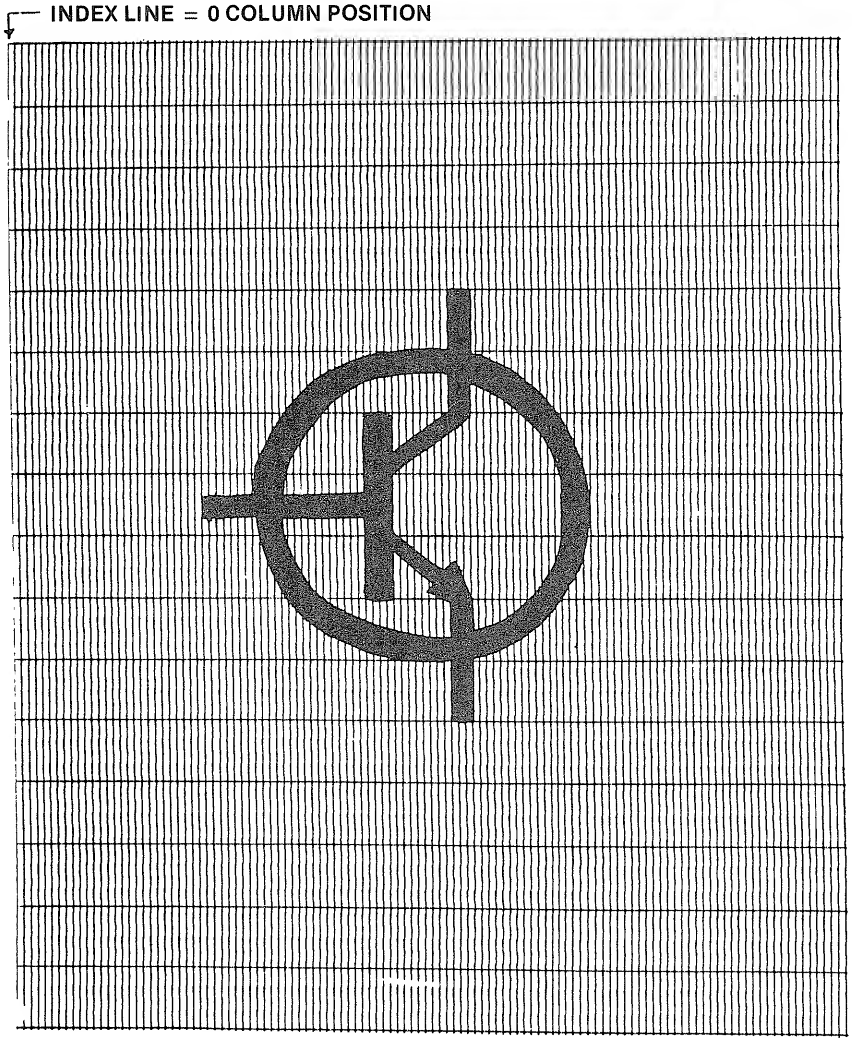


Figure 12-8. Laying Out the Figure

having to print large amounts of white space. The column position is specified by a CHR$(27);CHR$(16);CHR$(MM);CHR$(LL) code sequence, where MM is the quotient of the column position divided by 256, and LL is the remainder. The column position, by the way, is from the left margin position on the paper, which might *not* correspond to the left index column in this picture. Here we'll assume that the left margin of the paper corresponds to the left margin of the layout sheet.

Note: Because the repeat sequence contains numeric values that may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, identical to the top-of-form character, the print driver will attempt to execute a page eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line-feed character, it will be changed to a 13 in the print driver. Avoid repeat sequences containing these values by using a slightly

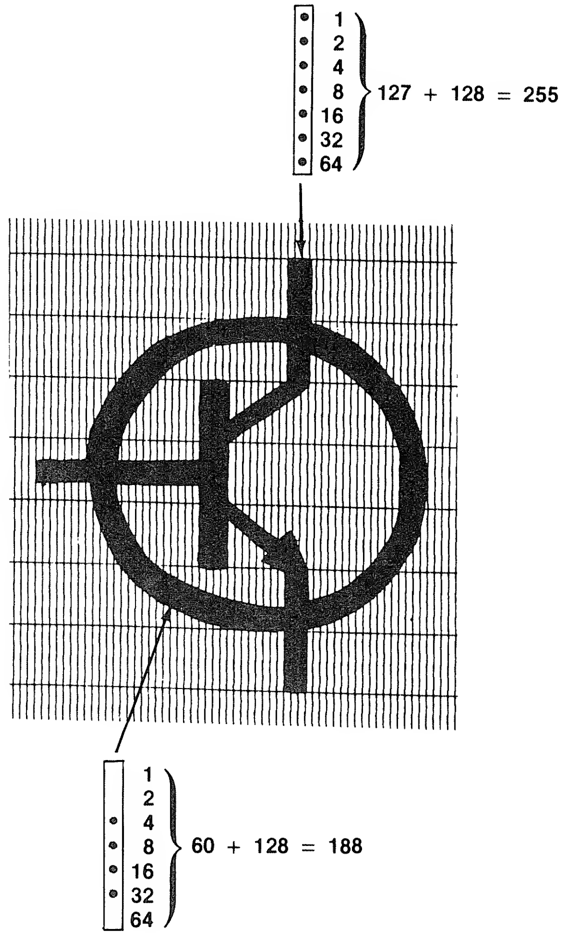


Figure 12-9. Calculating Graphics Character Weights

larger or smaller value in the CHR$() numeric variable, or use a combination of two or more escape sequences to get the same result. The STRING$ function in BASIC is also a good alternative.

Next, collect all of the CHR$() values up to the last blank area on a line. The result is shown in Table 12-1. There are a lot of terms (especially "leading" blank values of 128), because of the fine resolution that print graphics affords.

The CHR$() values can now be entered into a program after an LPRINT (PRINT#-2,) for each graphics line. (Of course, first you'll have to set graphics mode by a CHR$(18).) An alternative way to print the CHR$ values is to do something like what is shown in Listing 12-2.

This program lists all of the CHR$() values in DATA statements. This saves a lot of writing because the CHR$() characters do not have to be repeated. The carriage returns at the end of each line are included in the DATA statements as CHR$(13). To print the figure, a loop is done that READs a single DATA value and then prints it out as a CHR$(); value. The loop is ended when a final (dummy) value of -1 is READ by the loop. A -2 value in the DATA statements indicates a repeat code. We're using a STRING$ function here for

Table 12-1. Graphics codes for picture

LINE 1
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,255, 255, 255
LINE 2
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,192,224,
240,240,248,184,188,156,158,142,142,143,143,135,135,135,135,135,135,135,135,135,
135,255,255,255,143,142,142,158,156,188,184,248,240,240,224,192
LINE 3
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,128,128,128,128,128,128,128,128,128,128,128,128,224,248,252,254,191,143,135,
131,129,128,128,128,128,128,128,128,255,255,255,255,176,176,152,152,140,140,134,
134,131,131,129,128,128,128,128,128,128,128,128,129,131,135,143,191,254,252,248,
224
LINE 4
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,128,128,128,128,128,156,156,156,156,156,156,156,255,255,255,255,156,156,156,
156,156,156,156,156,156,156,156,156,255,255,255,255,128,128,128,128,128,128,128,
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,255,255,255,
255
LINE 5
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,128,128,128,128,128,128,128,128,128,128,128,128,131,143,159,191,254,248,240,
224,192,128,128,128,128,128,128,128,255,255,255,255,134,134,140,140,152,248,240,
240,248,248,192,128,128,128,128,128,128,128,128,192,224,240,248,254,191,159,143,
131
LINE 6
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,129,131,
135,135,143,142,158,156,188,184,184,248,248,240,240,240,240,240,240,240,240,240,
240,255,255,255,248,184,184,188,156,158,142,143,135,135,131,129
LINE 7
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
128,255,255,255

a repeat. When the program finds a -2 in a DATA value, it READs the next two values and uses them in a following STRING$ to repeat. The result of this printing is shown in Figure 12-10, an actual printout on the DMP-400. Printouts for other printers may have slightly different proportions.

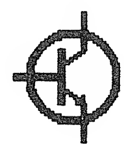


Figure 12-10. Graphics Example 1

The process above can be repeated for any pattern or picture to be printed. As you can see, however, it is laborious to encode the picture into the proper CHR$() values and it takes many values to produce a picture. If you are performing many such printouts, a good alternative is to buy a Color Computer and use it to generate the image before printing out the screen.

There are currently only a few programs that will manipulate printer graphics in the same way that the Color Computer does via its LINE, CIRCLE, and other commands.

As another example of graphics, look at the program in Listing 12-3. This cowboy program prints the graphics pattern shown in Figure 12-11 on a DMP-2100. (You can use the same figure on another printer, but it may appear in different proportions.)

The program reads DATA values one at a time, printing each as a graphics column, as in the previous program. The exceptions to this are the -1 value, which marks the end, and the -2 values, which mark the beginning of repeat data. The Repeat data is done as a STRING$, as in the previous program.

```
100 ' TRANSISTOR GRAPHICS PROGRAM
110 CLEAR 1000
120 LPRINT CHR$(18)
130 READ A
140 IF A=-1 THEN 190
150 IF A<>-2 THEN LPRINT CHR$(A);: GOTO 130
160 READ B,C
170 LPRINT STRING$(B,CHR$(C));
180 GOTO 130
190 LPRINT CHR$(30)
200 'TRANSISTOR DATA VALUES
210 DATA -2,61,128,255,255,255,13
220 DATA -2,38,128,192,224,240,240,248,184,188,156,158,142,142,143,143
230 DATA -2,10,135,255,255,255,143,142,142,158,156,188,184,248
240 DATA 240,240,224,192,13
250 DATA -2,33,128,224,248,252,254,191,143,135,131,129,-2,7,128,-2,4,255
260 DATA 176,176,152,152,140,140,134,134,131,131,129,-2,8,128,129,131
270 DATA 135,143,191,254,252,248,224,13
280 DATA -2,26,128,-2,7,156,-2,4,255,-2,12,156,-2,4,255,-2,24,128
290 DATA -2,4,255,13
300 DATA -2,33,128,131,143,159,191,254,248,240,224,192,-2,7,128,-2,4,255
310 DATA 134,134,140,140,152,248,240,240,248,248,192,-2,8,128,192,224
320 DATA 240,248,254,191,159,143,131,13
330 DATA -2,38,128,129,131,135,135,143,142,158,156,188,184,184,248,248
340 DATA -2,10,240,255,255,255,248,184,184,188,156,158,142,143,135
350 DATA 135,131,129,13
360 DATA -2,61,128,255,255,255,13
370 DATA -1
```

Listing 12-2. Transistor Graphics Program

```
100 'COWBOY GRAPHICS PROGRAM
110 CLEAR 2000
120 LPRINT CHR$(18)
130 READ A
140 IF A=-1 THEN 190
150 IF A<>-2 THEN LPRINT CHR$(A);: GOTO 130
160 READ B,C
170 LPRINT STRING$(B,CHR$(C));
180 GOTO 130
190 LPRINT CHR$(30)
200 'COWBOY VALUES
210 DATA -2,50,128,-2,11,128,224,252,140,-2,4,128,-2,3,192,13
220 DATA -2,50,128,-2,10,128,142,159,191,248,240,242,246,246,255,255,223,134,134
,130,13
230 DATA -2,50,128,-2,13,128,248,-2,3,255,159,143,159,190,240,224,-2,4,128,192,2
24,192,224,224,192,224,192,192,13
240 DATA -2,50,128,-2,11,128,176,-2,6,255,254,254,255,255,252,253,254,-2,6,255,1
91,-2,3,159,-2,3,255,190,252,168,13
250 DATA -2,50,128,-2,7,128,224,240,248,252,254,-2,18,255,224,-2,3,192,128,128,1
29,159,143,159,13
260 DATA -2,50,128,240,248,252,188,142,130,159,191,-2,6,255,159,191,255,255,159,
131,-2,4,129,128,128,129,129,131,135
270 DATA 252,152,128,129,131,198,252,13
280 DATA -2,50,128,129,135,159,252,-2,5,128,243,255,143,131,129,128,135,143,-2,3
,136,152,240,224,-2,5,128,140,142,135,-2,3,128
290 DATA 131,131,129,13
300 DATA -2,50,128,-2,10,128,129,131,134,140,184,240,224,13
310 DATA -1
```

Listing 12-3. Cowboy Graphics Program

PLOTTING ALONG WITH DOT-MATRIX AND DAISY-WHEEL PRINTERS

Many Radio Shack printers, including the daisy-wheel printers, have the ability to *back up* the paper (typesetters call this "reverse leading"). Generally, you won't want to use this feature to print graphics that require a great deal of precise repositioning.

To see whether your printer is capable of repositioning precisely, try this program. It sets word-processing mode, prints a plus/minus pattern, spaces eight inches, spaces back to the original line by using half line feeds, and then prints the pattern again. If your printer can precisely position the paper, the pattern will appear exactly overlaid.

```
100 LPRINT CHR$(20)
110 LPRINT"+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+-+-+-+-+-+-+-+-+-+-+-"
120 FOR I=1 TO 48
130 LPRINT CHR$(27);CHR$(30);
140 NEXT I
160 FOR I=1 TO 96
170 LPRINT
180 NEXT I
190 LPRINT"+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+-+-+-+-+-+-+-+-+-+-+-"
```

Figure 12-11. Graphics Example 2

Most likely, the repositioning will not be precise. You may be able to adjust the paper tension on a tractor feed or manually to get a better repositioning, but in general you should probably avoid using this technique for graphics. Typical results are shown in Figure 12-12.

GOOD REPOSITIONING

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

TYPICAL REPOSITIONING

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

Figure 12-12. Repositioning Example

We've been using a top-to-bottom approach on many of the graphics programs here because of the repositioning limitations. An exception to this is occasionally plotting, which prints points on a graph. The requirements here are not as stringent as repositioning a graphics dot within 1/120 of an inch over 8 inches—if you're off by a small amount, who's going to tell?

The program shown in Listing 12-4 uses the reverse leading capability of a dot-matrix or daisy-wheel printer to plot points on a graph. A typical plot from the program is shown in Figure 12-13, which shows a plot of a cosine wave.

The program is defined as a subroutine that you can call from your own

```
10000 'SUB GRAPH (MAIN)
10010 ZX=INT(ZX) : ZY=-INT(ZY)
10020 IF ZX=ZA AND ZY=ZB THEN RETURN
10030 IF ZX>ZW/2 OR ZX<-ZW/2 OR ZY>ZH/2 OR ZY<-ZH/2 THEN RETURN
10040 IF ZX=ZA THEN 10060
10050 ZN=ZX-ZA : ZA=ZX : GOSUB 10090
10060 IF ZY=ZB THEN 10080
10070 ZN=ZY-ZB : ZB=ZY : GOSUB 10220
10080 ZZ$="."+ZB$ : FOR ZT=1 TO 0 : NEXT ZT : GOSUB 10330 : RETURN
10090 'SUB MOVE HORIZ (IN 1/120'S)
10100 PRINT"horiz:";ZN
10110 IF ZN<0 THEN 10170
10120 ZR=ZN-INT(ZN/ZC)*ZC
10130 ZN=INT(ZN/ZC)
10140 ZZ$=STRING$(ZN,32) : GOSUB 10330
10150 IF ZR>0 THEN FOR ZT=1 TO ZR : ZZ$=ZS$ : GOSUB 10330 : NEXT ZT
10160 RETURN
10170 ZN=-ZN
10180 ZR=ZN-INT(ZN/9)*9 : ZN=INT(ZN/9)
10190 ZZ$="" : IF ZN>0 THEN FOR ZT=1 TO ZN : ZZ$=ZZ$+CHR$(8)+CHR$(9) : NEXT
 ZT
10200 ZZ$=ZZ$+CHR$(8)+CHR$(ZR) : GOSUB 10330
10210 RETURN
10220 'SUB MOVE VERT (IN 1/48'S)
10230 PRINT"vert:";ZN
10240 IF ZN<0 THEN 10290
10250 ZR=ZN-INT(ZN/ZF)*ZF : ZN=INT(ZN/ZF)
10260 IF ZN>0 THEN ZZ$="" : FOR ZT=1 TO ZN : ZZ$=ZZ$+ZM$ : NEXT ZT : GOSUB
10330
10270 IF ZR>0 THEN ZZ$="" : FOR ZT=1 TO ZR : ZZ$=ZZ$+ZL$ : NEXT ZT : GOSUB
10330
10280 RETURN
10290 ZN=-ZN : ZR=ZN-INT(ZN/ZF)*ZF : ZN=INT(ZN/ZF)
10300 FOR ZT=1 TO ZN+1 : ZZ$=ZR$ : GOSUB 10330  : NEXT ZT
10310 FOR ZT=1 TO ZF-ZR : ZZ$=ZL$ : GOSUB 10330  : NEXT ZT
10320 RETURN
10330 'SUB PRINT STRING
10340 LPRINT ZZ$; : POKE 16426,5 : RETURN
10350 'SUB INITIALIZE
10360 '*** CHANGE ZS$=CHR$(27)+CHR$(0) FOR DWP-410
10370 '*** CHANGE ZF=20, ZC=18, ZL$=CHR$(27)+CHR$(49) FOR DMP-2100
10380 '*** CHANGE ZF=12, ZL$=CHR$(27)+CHR$(50) FOR DMP-200
10390 ZC=12                                      ' CHAR WIDTH
10400 ZB$=CHR$(8)+CHR$(ZC/2)+CHR$(8)+CHR$(ZC/2)  ' FULL BACKSPACE
10410 ZS$=CHR$(27)+CHR$(1)                       ' 1/120 IN. SPACE
10420 ZR$=CHR$(27)+CHR$(30)+CHR$(27)+CHR$(30)    ' FULL REV. LF
10430 ZL$=CHR$(27)+CHR$(26)                      ' 1/48 IN. LF
10440 ZM$=CHR$(27)+CHR$(28)+CHR$(27)+CHR$(28)    ' FULL LF
10450 ZF=8                                       ' FRACT. OF LF
10460 ZN=ZH/2 : GOSUB 10220
10470 ZN=ZW/2 : GOSUB 10090
10480 ZA=0 : ZB=0 : RETURN
10490 END
```

Listing 12-4. Graph Plotter Program

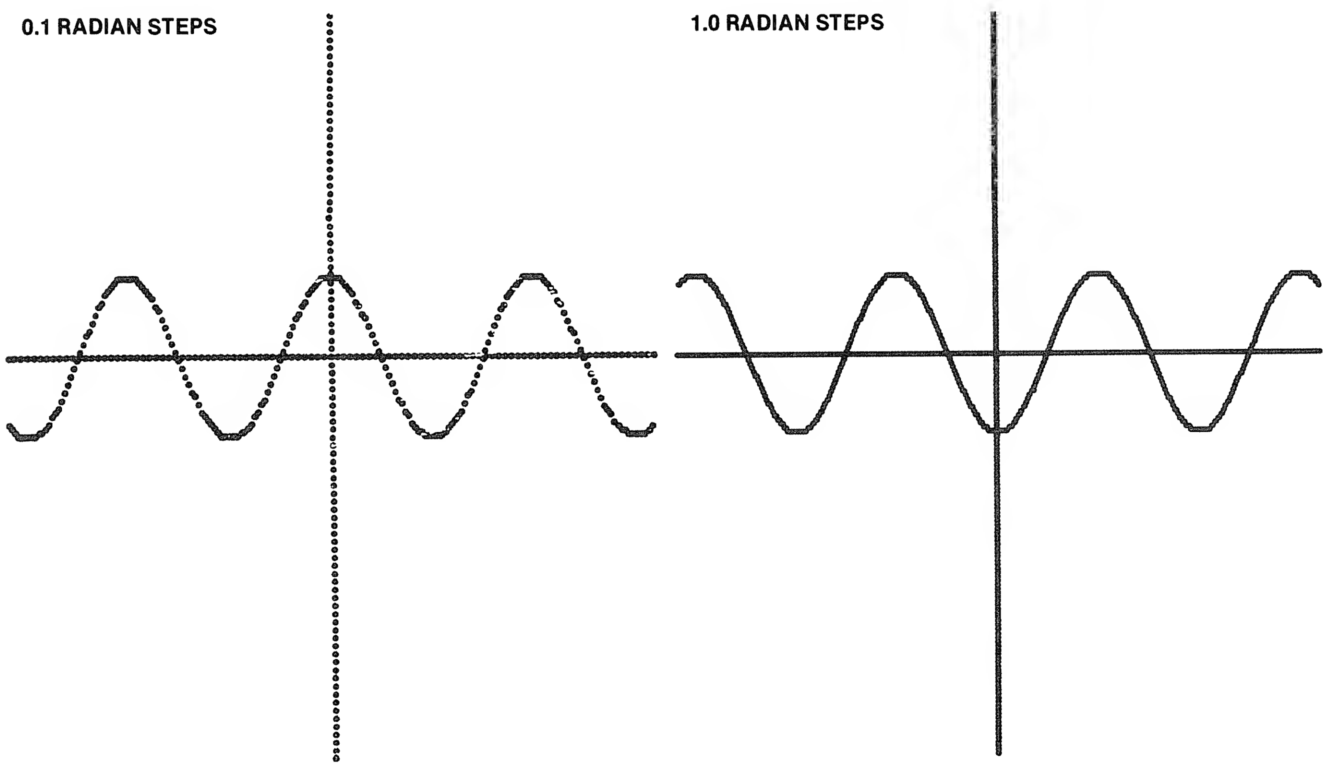


Figure 12-13. Plot Example

```
100 'PRINTER GRAPHICS SUBROUTINE
110 CLEAR 1000
120 ZW=400 : ZH=200
130 GOSUB 10350
140 FOR X=-200 TO 200 STEP 4 : ZX=X : ZY=0 : GOSUB 10000 : NEXT X
150 FOR Y=-100 TO 100 STEP 2: ZY=Y : ZX=0 : GOSUB 10000 : NEXT Y
160 FOR X=-10 TO 10 STEP .1
170 ZX=X*20 : ZY=COS(X)*20 : GOSUB 10000
180 NEXT X : STOP
```

Listing 12-5. Graph Plotter Program Calling Sequence

BASIC program. The call for the cosine plot is shown in Listing 12-5. There are four parameters involved:

1. ZW is the width of the plot in microspaces (daisy-wheel printers) or dot positions (dot-matrix printers). This will be the actual physical width of the plot.
2. ZH is the height of the plot in increments of 1/48 of an inch (daisy-wheel printers) or 1/120 of an inch (dot-matrix printers). This will be the actual physical height of the plot.
3. ZX and ZY are the coordinates for the point on the graph in horizontal and vertical units.

The width in ZW and the height in ZH are defined only once in the "calling" program. A call is made to the subroutine at 10350 with ZW and ZH defined. This subroutine sets up the coordinates the graph will use. The graph is assumed to be in standard Cartesian coordinates, with an origin in the center. The maximum and minimum X and Y will be one-half of the ZW and ZH values, as shown in the figure.

After the coordinates are established by the 10350 subroutine, the remaining action will be taken by the subroutine starting at line 10000. This subroutine is called with the point to be plotted in ZX and ZY. The subroutine will move the paper in the printer, print a period for the point, and then return. Every point to be plotted requires a separate call to the 10000 subroutine.

Before running the program, make these changes, depending upon the type of printer you have:

- DW-IIB, DWP-210: Leave as is
- DWP-410: Change line 10410 to

```
10410 ZS$=CHR$(27);CHR$(0)
```

- DMP-400, -420, -500: Leave as is
- DMP-200: Change lines 10450 and 10430 to

```
10430 ZL$=CHR$(27)+CHR$(50)
10450 ZF=12
```

- DMP-2100: Change lines 10430 and 10450 to

```
10430 ZL$=CHR$(27)+CHR$(50)
10450 ZF=20
```

Printer Hint

HOW THE PLOTTING PROGRAM WORKS

The plotting (graph) program moves the paper in a daisy-wheel or dot-matrix printer to the proper line and microspace or dot column position for each point plotted. The secret in the program is to avoid the dreaded full reverse line feed and backspace sequences which might have a CHR$(10) or CHR$(12) in them. These characters will create havoc in the BASIC print driver. To avoid problems, the program spaces back in half-line spaces, and moves forward in 1/48-inch increments for daisy-wheels. For dot-matrix printers, the program spaces back in half-line spaces and moves forward in 1/120-inch and 1/72-inch increments.

Horizontally, the program moves in 1/60- or 1/120-inch increments for daisy-wheel printers, and in single dot-columns for dot-matrix printers.

A single period is printed when the proper point is found. Not too elegant, but the program can create some nice graphs!

DMP-2100 HIGH-RESOLUTION GRAPHICS

The DMP-2100 is an interesting printer because it has a 24-pin(!) print head. The 24 pins occupy about the same space as seven pins on a normal print head on printers like the DMP-500. The DMP-2100 uses all 24 pins to produce different fonts, or typefaces, and the result is dot-matrix printing which looks remarkably like daisy-wheel letter quality printing. (When the printed copy is photocopied, the dots blend in even more and the quality is further improved.)

A total of 24 pins in a vertical printing column 0.12 inches high gives a dot resolution of 200 dots per vertical inch, or 2000 dots in a 10-inch vertical

printing area on a piece of paper. What about the horizontal resolution? Each dot printed in any of the other modes is actually made up of a three-by-three matrix of points in high-resolution mode. This means that there are actually three points for every dot horizontally and since there are 60 dots per inch normally, there will be 180 dots per inch in high-resolution mode, or about 1800 dots across a 10-inch width of paper, as shown in Figure 12-14. There are nine points for every normal dot. Putting it another way, there are 3,600,000 points in a 10-inch by 10-inch print area on the paper, and each one of those points can be programmed to print!

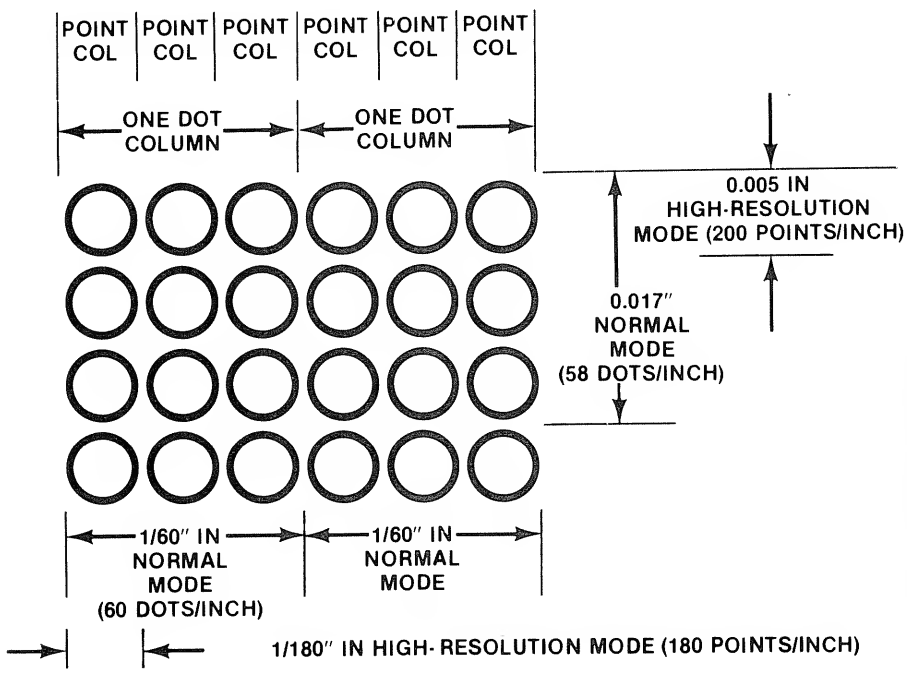


Figure 12-14. DMP-2100 Printing

Although there's a great deal of capability here, don't forget that processing a lot of points takes a large amount of time, especially if you are working in BASIC. A typical program loop of

```
100 FOR I = 1 TO 1000
110 NEXT I
```

takes less than two seconds on a typical Radio Shack system (much faster on the Tandy 2000), making the processing time per *iteration* about 2/1000 of a second, and that's for the simplest loop. Assuming that we could maintain this speed while processing points for the printer, we'd require 2/1000 times 3,600,000, or 7,200 seconds, to process a 10-by-10-inch area in high-resolution mode. That's about two hours! In fact, you'd expect to spend from three to ten times longer than that to process the points using BASIC, and that makes casual processing in high-resolution mode prohibitive. However, high-resolution mode can be used for smaller areas, and for such things as creating your own fonts (see "Designing Your Own Character Sets"). Another option is to use assembly language to speed up high-

Printer Hint

ASSEMBLY LANAGUAGE—AGAIN

Not to harp too much on the subject, but you might want to consider assembly language programming. On the negative side, assembly language is very tedious and hard to learn. However, once you learn it, you can do some truly amazing things and at very high speeds—up to hundreds of times faster than BASIC. The DMP-2100, with its extraordinarily high point density, is a perfect candidate for a high-speed assembly language program. You can cut down the programming overhead time drastically, and utilize the DMP-2100 to its best advantage. There are two courses available from Radio Shack for assembly language—"The Assembly Language Tutor for the Model I, III, and 4" and "The Assembly Language Tutor for the Color Computer." Each is a good starting point for assembly language studies.

resolution mode and other printing—you could speed up BASIC processing by a factor of 100 or more!

USING HIGH-RESOLUTION GRAPHICS

You can use high-resolution graphics mode in any text-printing mode. That's nice because it allows you to intersperse text with your own high-resolution characters. There's one primary escape sequence for using high-resolution graphics and it looks like this:

```
CHR$(27);CHR$(73);CHR$(MM);CHR$(LL);CHR$(XX); . . . CHR$(XX);
```

The escape sequence is divided into three parts. The CHR$(27) and CHR$(73) inform the DMP-2100 that the data following will be in high-resolution mode.

The CHR$(MM) and CHR$(LL) are similar to the dot-column position sequence for normal graphics or text—the MM in this case, however, is *high order* value for the *point-column length*, and the LL is the *low order* value. The *point column* is the dot column times three, because there are three horizontal points for every horizontal dot, as shown in Figure 12-15. Note that the point column *length* is different from the point-column *position*. This will be the actual number of point columns printed for the graphics, as shown in the figure. The MM and LL values are computed the same way as for the dot-column values—the point-column number is divided by 256, with the remainder going to LL, and the quotient going to MM. A total of 1000 point columns, for example, would use an MM value of 3 (1000/256 is a quotient of 3), and an LL value of 232 (1000/256 = 3, remainder of 232).

The last part of the escape sequence consists of a series of data values. Each data value corresponds to one 8-pin column print. Since there are three 8-pin columns making up each 24-pin print, there will be three times the number of data values as there are point columns. If the point-position length specifies 24 point columns to be printed, there are 72 data values. Until the printer receives the complete set of data values, it treats the next data value as part of the high-resolution graphics data, so it's very important to send the correct number of data values, after the number of point columns are specified.

Note: Because this escape sequence contains numeric values that may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, identical to the top-of-form character, the print driver will attempt to execute a page eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line-feed character, it will be changed to a 13 in the print driver. You must avoid these values if possible. About the only other alternative you have is to circumvent the BASIC printer driver by using your own print driver. Because the dot density is so great in the DMP-2100, it should be possible to set a dot adjacent to the proper one without affecting the graphics too dramatically.

Let's take a concrete example. Suppose that you wanted to print the shape shown in Figure 12-16. This shape consists of 24 points on the left edge, two columns of intermediate points, and 24 points along the right edge—a kind of abbreviated "A." We'd reproduce the actual printing here, but it's very narrow!

SETS HIGH-RES SETS # OF POINT COLUMNS

```
CHR$(27);CHR$(73);CHR$(0);CHR$(3);
CHR$(112);CHR$(128);CHR$(240);
CHR$(96);CHR$(16);CHR$(48);
CHR$(64);CHR$(1);CHR$(224)
```

3 VALUES FOR EVERY POINT COLUMN

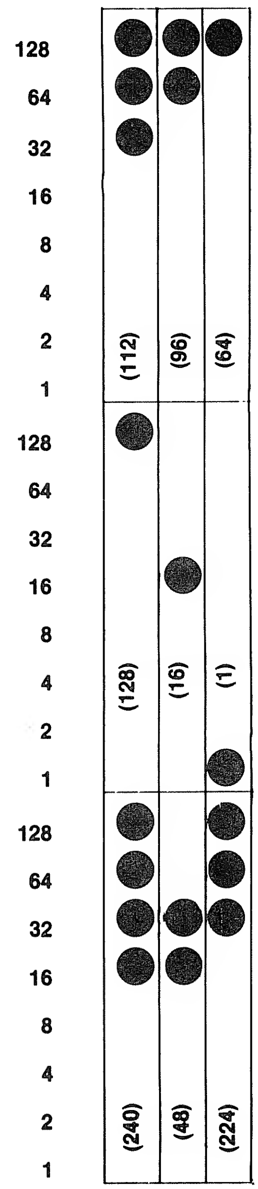


Figure 12-15. DMP-2100 High-Resolution Printing

There are four point-columns to be printed here—a little bit wider than a single graphics dot in normal graphics mode. The point position length is therefore four, and we'll have CHR$(27);CHR$(73); CHR$(0);CHR$(4) as the first part of the escape sequence.

Each point column is divided into three segments, as shown in the figure,

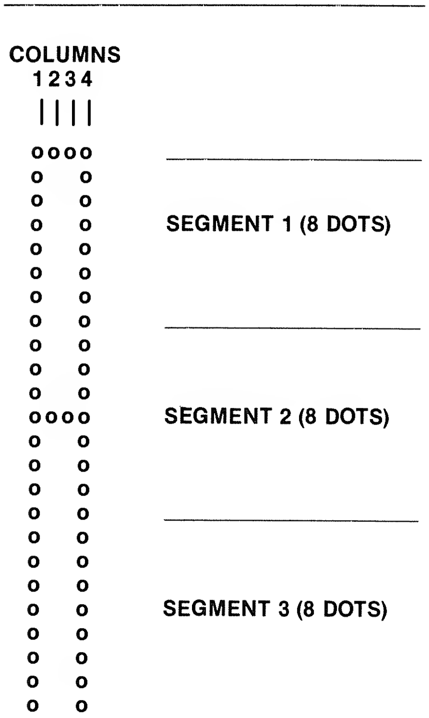


Figure 12-16. Simple Shape for DMP-2100 High Resolution Printing

with each segment specifying eight points. Like the seven-dot graphics values, each point position has a weighted value. *Unlike* the seven-dot graphics values, though, the topmost point is 128, the next 64, the next 32, the next 16, the next 8, the next 4, the next 2, and the last is 1, as shown in Figure 12-17.

Since all pins are "on" for the first segment, the value here is 255 (128 + 64 + 32 + 16 + 8 + 4 + 2 + 1). The next value (the second segment down in the first point column) is also 255, and the last is also 255. The first three CHR$() values for the points are, therefore,CHR$(255);CHR$(255);CHR$(255);.

The topmost segment of the next point column has only the 128-weighted pin "on," so the data value is 128. The middle segment has only the 16-weighted pin "on," so the value is 16. The bottom segment has no pins "on," so the value is 0. The next three CHR$() values are, therefore, CHR$(128);CHR$(16);CHR$(0);

The third point column is identical to the second, and the next three values will be the same: CHR$(128);CHR$(16);CHR$(0);.

The final point column is the same as the first, CHR$(255);CHR$(255); CHR$(255);.

The complete code to print this shape is:

```
100 LPRINT CHR$(27);CHR$(73);CHR$(0);CHR$(4);
CHR$(255);CHR$(255);CHR$(255);CHR$(128);CHR$(16);
110 LPRINT CHR$(0);CHR$(128);CHR$(16);CHR$(0);
CHR$(255);CHR$(255);CHR$(255)
```

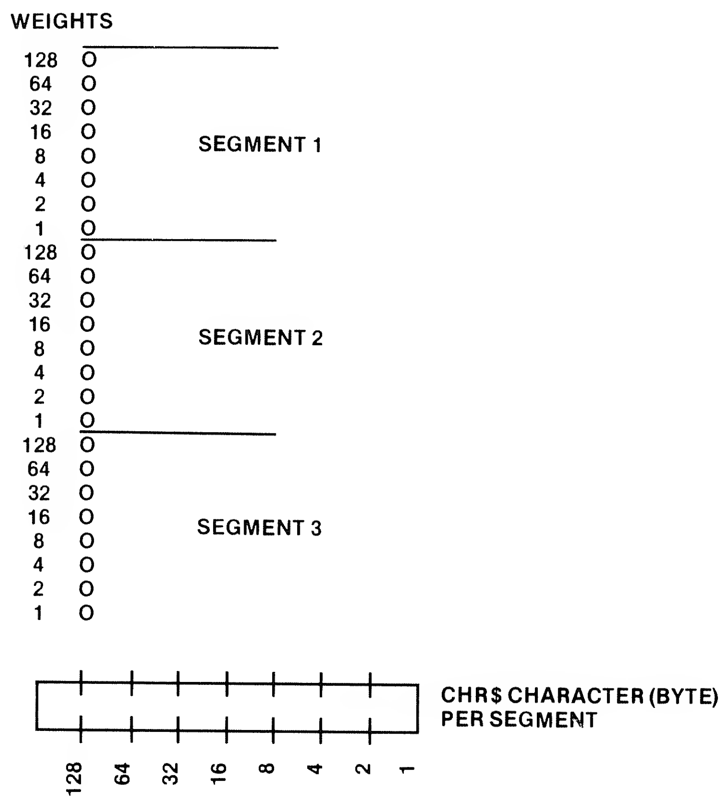


Figure 12-17. DMP-2100 Point Weights

HIGH-RESOLUTION MODE AND LINE FEEDS

To print consecutive rows of high-resolution graphics, use the 4/5-line feed code—CHR$(27);CHR$(71). This escape sequence will set a 4/5-line feed that is 4/5 of the nominal six-lines-per-inch spacing (0.133 inch). There will be no white space between adjacent high-resolution rows. To print a checkerboard pattern of high-resolution dots over an area 2.66 inches wide by 1.06 inch high, for example, you'd do something like this:

```
100 FOR J=1 TO 8
110 LPRINT CHR$(27);CHR$(73);CHR$(1);CHR$(224);
120 FOR I=1 TO 480
130 LPRINT CHR$(170);CHR$(170);CHR$(170);
140 NEXT I
150 LPRINT CHR$(27);CHR$(71);
160 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(0);
170 NEXT J
```

The escape sequence at line 160 repositions the print head at the beginning of the next line. The 4/5-line feed is one of those line feeds that are acted upon immediately, causing vertical movement, but not a carriage return. If this escape sequence is not used, the next row of graphics starts displaced to the right by 480 point-positions!

See Figure 12-18 for the actual result.

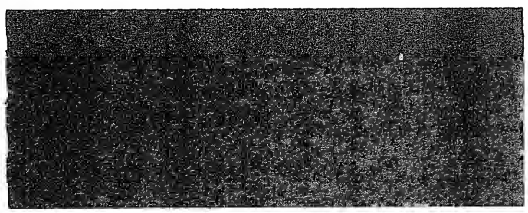


Figure 12-18. DMP-2100 Checkboard Pattern

Note: Avoid printing points in the same column. Stagger the dots instead, as shown in Figure 12-19. We violated this rule in our short code segment which printed out the special symbol with two vertical lines, but do as we say and not as we do. This is an electrical limitation and should be observed during all high-resolution printing. The graphics resolution is dense enough so that it's virtually impossible to see that solid characters are actually printed with this checkerboard pattern, as you can tell from the program above if you actually print the pattern.

For details on how to use the higher-resolution graphics mode to create your own fonts and type faces, see "Redefining the Character Set or Designing Your Own Characters" in this chapter.

DMP-110 HIGH-RESOLUTION GRAPHICS

The DMP-110 has a high-resolution graphics mode that allows you to print 16 dots in a vertical column that is the same height as a normal graphics

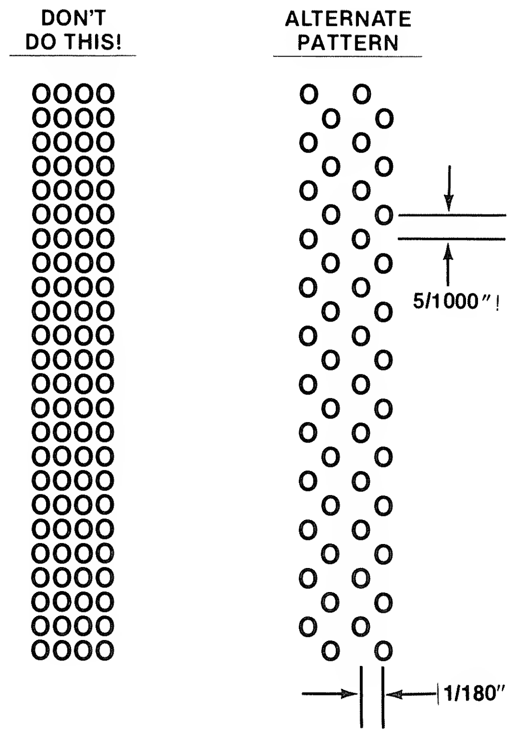


Figure 12-19. Staggering Points on the DMP-2100

column—about 0.1 inch. The DMP-110 is capable of the high-resolution graphics because of its unique print mechanics. Rather than firing a matrix of pins, the DMP-110 uses two print hammers to strike a ridged platen at precise times to form the print dots, as shown in Figure 12-20.

A total of 16 dots in a vertical printing column 0.1 inch high gives a dot resolution of 160 dots per vertical inch, or 1600 dots in a 10-inch vertical printing area on a piece of paper. For horizontal resolution, there are 960 dot positions per line, or 120 dots per inch. This means there are 1,536,000 dots in an 8-inch by 10-inch print area on the paper, and each one of those points can be programmed to print!

Although that's a great deal of capability, don't forget that processing a lot of points takes a great deal of time, especially if you are working in BASIC. A typical program loop of

```
100 FOR I=1 TO 1000
110 NEXT I
```

takes less than two seconds on a typical Radio Shack system, making the processing time per iteration about 2/1000 of a second, and that's for the simplest loop. Assuming that we could maintain this speed while processing points for the printer, we'd require 2/1000 times 1,500,000, or 3,000 seconds, to process an 8-by-10-inch area in high-resolution mode. That's about 50 minutes! In fact, you'd expect to spend from three to ten times longer than that to process the points using BASIC, and that makes a casual processing in high-resolution mode prohibitive. However, high-resolution mode can be used for smaller areas, and for such things as creating your

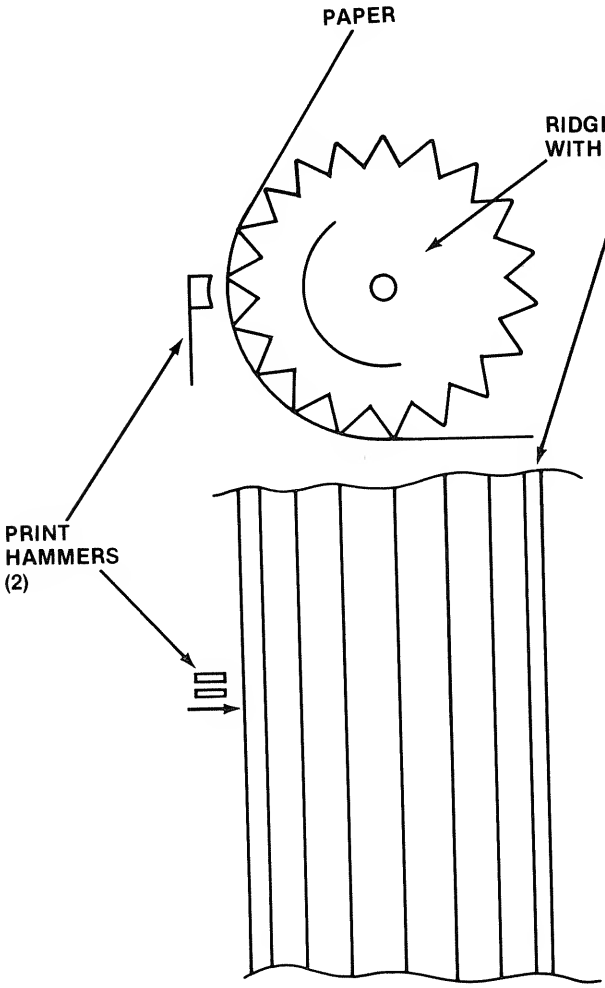


Figure 12-20. DMP-110 Printing

own fonts (see "Redefining the Character Set or Designing Your Own Characters"). Another option is to use assembly language to speed up high-resolution mode and other printing—you could speed up BASIC processing by a factor of 100 or more!

USING HIGH-RESOLUTION GRAPHICS

You can use high-resolution graphics mode in any printing mode. That's nice because it allows you to intersperse text with your own high-resolution characters. There's one primary escape sequence for using high-resolution graphics and it looks like this:

CHR$(27);CHR$(73);CHR$(MM);CHR$(LL);CHR$(XX); . . . CHR$(XX);

The escape sequence is divided into three parts. The CHR$(27) and CHR$(73) inform the DMP-110 that the data following will be in high-resolution mode.

The CHR$(MM) and CHR$(LL) are similar to the dot column position

sequence for normal graphics or text—the MM in this case, however, is the high order value for the dot-column length, and the LL is the low order value for the dot-column length. Note that the dot-column length is different from the dot-column position. This will be the actual number of dot columns printed for the graphics, as shown in Figure 12-21. The MM and LL values are computed the same way as for normal graphics dot-column values—the dot-column number is divided by 256, with the remainder going to LL, and the quotient going to MM. A total of 900 dot columns, for example, would use an MM value of 3 (900/256 is a quotient of 3), and an LL value of 132 (900/256 = 3, remainder of 132).

The last part of the escape sequence consists of a series of data values. Each data value corresponds to one eight-dot column print. Since there are two eight-dot columns making up each 16-dot print, there will be twice the number of data values as there are dot columns. If the dot-position length specifies 20 dot columns to be printed, there are 40 data values. Until the printer receives the complete set of data values, it treats the next data value as part of the high-resolution graphics data, so it's very important to send the correct number of data values, after the number of dot columns are specified.

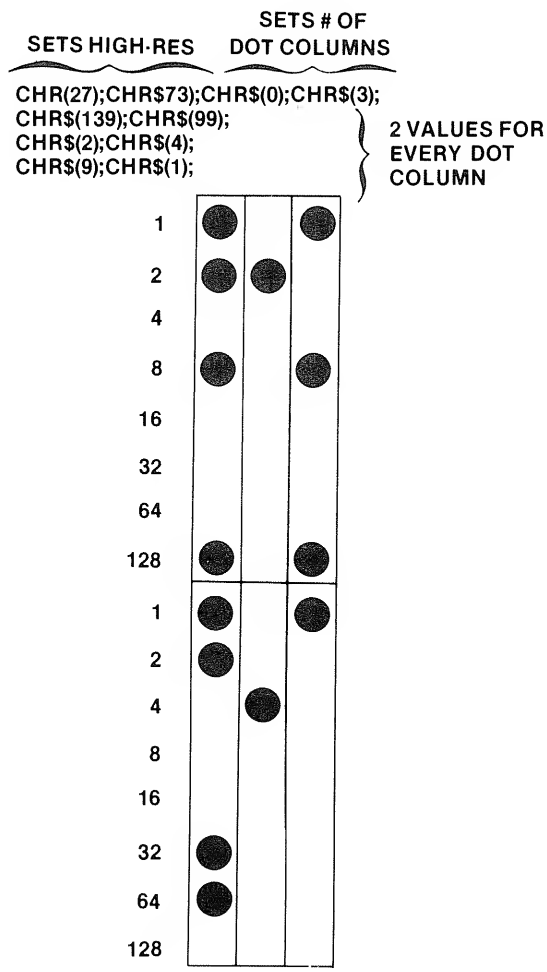


Figure 12-21. DMP-110 High-Resolution Printing

Note: Because this escape sequence contains numeric values that may be any number from 0 through 255, you may have problems in using the BASIC printer driver. If a numeric value in a CHR$() turns out to be a 12, identical to the top-of-form character, the print driver will attempt to execute a page eject by doing a series of new lines after it receives the 12. If a numeric value turns out to be a 10, identical to a line-feed character, it will be changed to a 13 in the print driver. Although it's hard to avoid these characters, it may be possible to set adjacent dots in graphics such that the graphics pattern isn't affected too dramatically.

For a concrete example, suppose that you wanted to print the shape shown in Figure 12-22. This shape consists of 16 points on the left edge, two intermediate points, and 16 points along the right edge—a kind of abbreviated "A."

There are four dot-columns to be printed here—a little narrower than a normal text character. The dot position length is, therefore, four; and we'll have CHR$(27);CHR$(73);CHR$(0);CHR$(4) as the first part of the escape sequence.

Each dot column is divided into two segments, as shown in the figure, with each segment specifying eight points. Like the seven-dot graphics values (see "Basics of Graphics Mode"), each point position has a weighted value—the topmost point is 1, the next 2, the next 4, the next 8, the next 16, the next 32, and the next 64, *but* (unlike other printers) the last is 128, as shown in Figure 12-23.

Since all pins are "on" for the first segment, the value here is 255 (1 + 2 + 4 + 8 + 16 + 32 + 64 + 128). The next value (the second segment down in the first point-column) is also 255. The first two CHR$() values for the points are,therefore, CHR$(255);CHR$(255);.

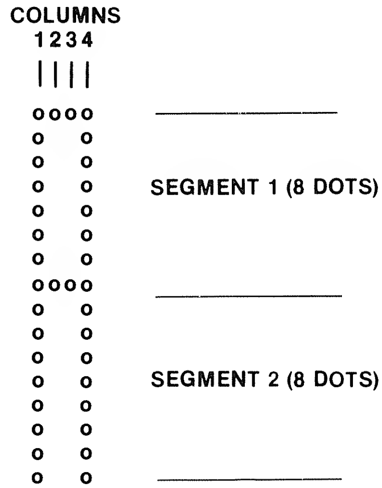


Figure 12-22. Simple Shape for DMP-110 High-Resolution

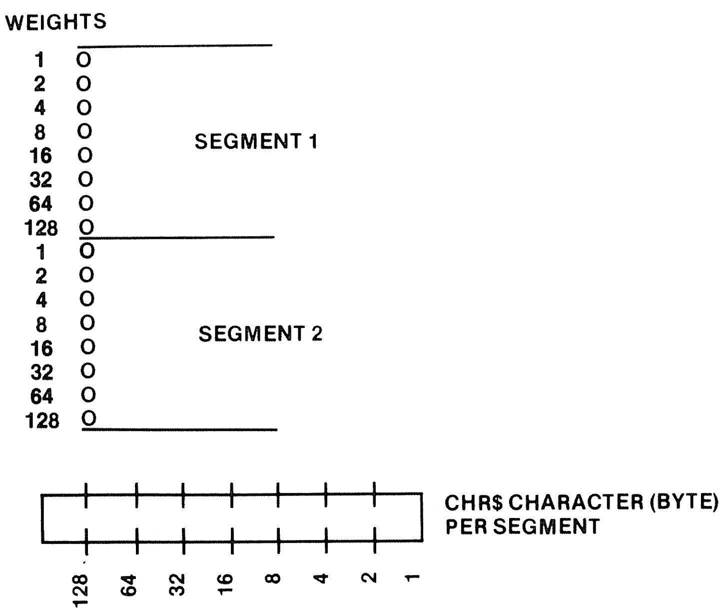


Figure 12-23. DMP-110 High-Resolution Weights

The topmost segment of the next point-column has the 1-weighted dot "on," so the data value is 1. The bottom segment has the 8-weighted dot "on", so the value is eight. The next two CHR$() values are therefore CHR$(0);CHR$(8);.

The third point-column is identical to the second, and the next two values will be the same — CHR$(0);CHR$(8);.

The final point-column is the same as the first — CHR$(255);CHR$(255);.

The complete code to print this shape is:

```
100 LPRINT CHR$(27);CHR$(73);CHR$(0);CHR$(4);
CHR$(255);CHR$(255);CHR$(0);CHR$(8);
110 LPRINT CHR$(0);CHR$(8);CHR$(255);CHR$(255);
```

HIGH-RESOLUTION MODE AND LINE FEEDS

To print consecutive rows of high-resolution graphics, use the 4/5 line feed code — CHR$(27);CHR$(71). This escape sequence will do a 4/5-line feed that is 4/5 of the nominal six-lines-per-inch spacing (4/5 of 0.166 inch). No white space will appear between adjacent high-resolution rows. To print a checkerboard pattern of high-resolution dots over an area four inches wide by 1.06 inches high, for example, you'd do something like this:

```
100 FOR J=1 TO 8
110 LPRINT CHR$(27);CHR$(73);CHR$(1);CHR$(224);
120 FOR I=1 TO 480
130 LPRINT CHR$(170);CHR$(170);
140 NEXT I
150 LPRINT CHR$(27);CHR$(71);
160 LPRINT CHR$(10);CHR$(16);CHR$(0);CHR$(0);
170 NEXT J
```

The escape sequence at line 160 repositions the print head at the beginning of the next line. If this is not used, the next row of graphics starts one line lower, but displaced to the right by 480 dot columns! The rationale for this is that the 4/5 line feed is one of those line spacing commands that is acted upon immediately to move the paper; but it does not cause a carriage return — only a line feed. See Figure 12-24 for the actual result.

For details on how to use the higher resolution graphics modes for creating your own fonts and type faces, see "Redefining the Character Set or Designing Your Own Characters" in this chapter.

INTERMIXING TEXT AND GRAPHICS

On most printers with graphics, you can intermix text and graphics by switching back and forth between text (data- and word-processing) and graphics modes. You could, for instance, print the figure shown in Figure 12-25 with the following program:

```
90 CLEAR 1000
100 LPRINT CHR$(18);CHR$(255);STRING$(120,CHR$(129));CHR$(255)
110 LPRINT CHR$(255);CHR$(30);"    TEXT IN BOX    ";
CHR$(18);CHR$(255)
120 LPRINT CHR$(255);STRING$(120,CHR$(192));CHR$(255)
130 LPRINT CHR$(30)
```

Figure 12-24. DMP-110 High-Resolution Print Example

Figure 12-25. Intermixing Text and Graphics

However, in some cases you might want to stay in graphics mode and print text as part of the graphics, or in other cases you might want to print enlarged text, or your own fancy font or typefaces. We'll show you how to do all of those things here.

PRINTING TEXT IN GRAPHICS MODE

Text characters are defined in the built-in character sets of the printers as a dot-matrix. The dot-matrix ranges from a five-by-seven dot-matrix for older or less expensive printers, to a nine-by-twenty dot matrix for more recent printers. Newer printers also print in half-dot-columns so that adjacent columns can overlap. A typical character set definition is shown in Figure 12-26, which defines a portion of the character set for the DMP-120.

There's no reason that you can't use the same definition in graphics mode to print characters in the same way. The printing is much slower (if you're using BASIC), but you can integrate text with graphics.

The best way to handle this is with a *character generation subroutine*. Such a subroutine is shown in Listing 12-6.

The basic "skeleton" of the subroutine looks like this:

```
10 ' SUB PRINT ONE CHARACTER AS GRAPHICS DOT MATRIX
11 ' ZZ$=CHARACTER TO BE PRINTED
12
.
.
.
99 RETURN
```

The vertical dots stand for missing BASIC code. The subroutine is located starting at line 10 to make it faster (at the beginning of your program), and so you can add your own graphics program lines after it.

To use the subroutine at any time, just put the text to be printed into string variable ZZ$, call the subroutine, and the subroutine will print out the text from the graphics data. The subroutine can be called at any time and for

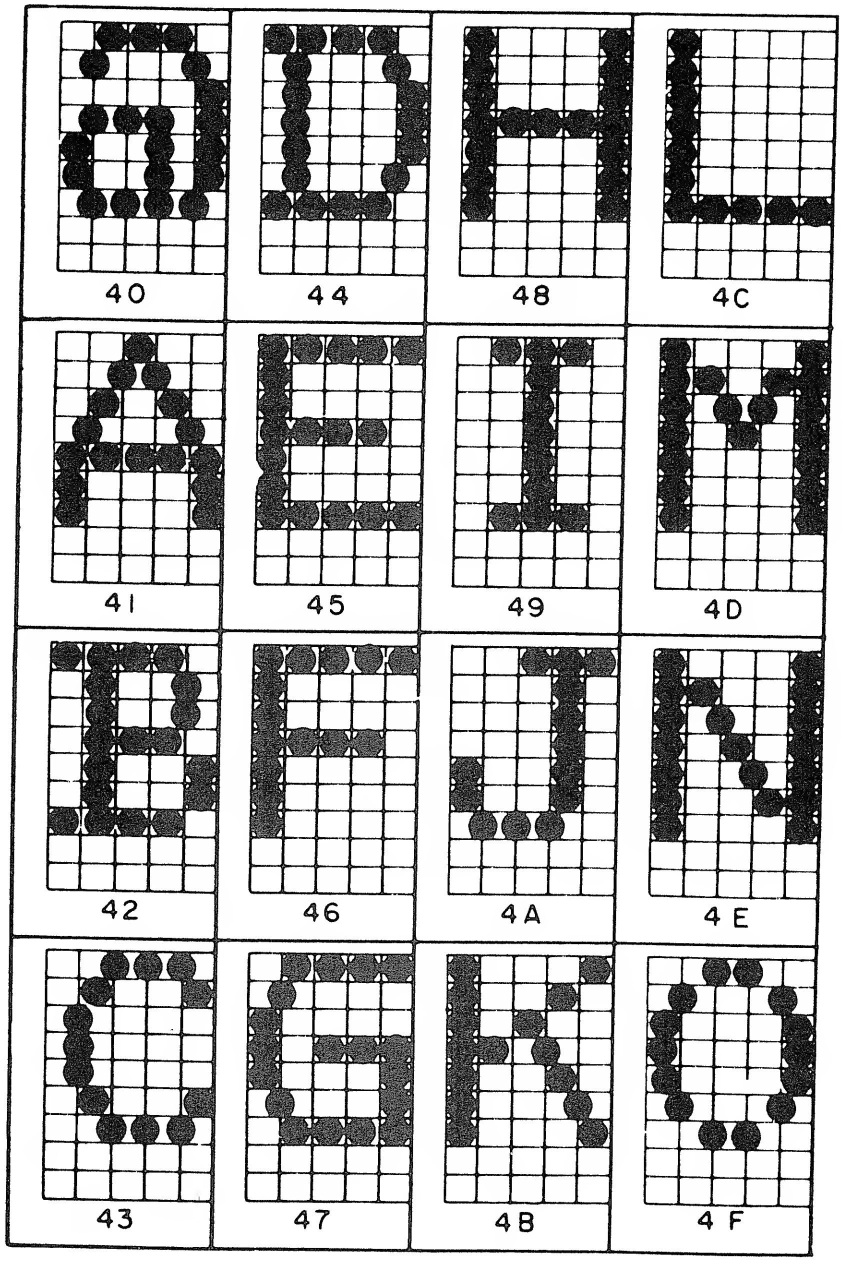


Figure 12-26. Typical Printer Character Set Definition

any position on the printout. It will blindly print the characters at that point on the listing. You'll have to make certain there's enough room remaining on the paper, and generally make certain that the layout is proper for the printing to proceed.

To call the subroutine to print "Text Sample", you'd do this:

```
100 ZZ$ = "Text Sample": GOSUB 10000
110 (more of your own graphics here)
```

Within the subroutine, characters are defined by a five-by-seven dot matrix. We could have used a larger matrix, but there's a problem when

```
10 'SUB 7 BY 10 PATTERN GENERATOR
11 CLEAR 2000
12 LPRINT CHR$(18)
13 DIM ZT(127,10)
14 ZZ$=CHR$(3)+CHR$(4)+CHR$(2)+CHR$(4)+CHR$(1)+CHR$(4)+CHR$(0)+CHR$(4)+CHR$(6)+C
HR$(4)+CHR$(5)
15 FOR ZI=0 TO 127
16 FOR ZJ=1 TO 10: ZT(ZI,ZJ)=128: NEXT ZJ
17 NEXT ZI
18 ZI=-1
19 IF ZI=>127 THEN 26
20 ZI=ZI+1: ZJ=0
21 IF ZJ=>10 THEN 19
22 ZJ=ZJ+1
23 READ ZT(ZI,ZJ)
24 IF ZT(ZI,ZJ)=-1 THEN 26
25 GOTO 21
26 ZL=LEN(ZZ$)
27 IF ZL=0 THEN 40
28 FOR ZX=1 TO ZL
29 ZC$=MID$(ZZ$,ZX,1)
30 ZN=ASC(ZC$)
31 IF ZN>-1 AND ZN<=ZL THEN 34
32 LPRINT STRING$(10,128);
33 GOTO 37
34 FOR ZY=1 TO 10
35 LPRINT CHR$(ZT(ZN,ZY));
36 NEXT ZY
37 LPRINT CHR$(128); 'SPACE AFTER CHAR
38 NEXT ZX
39 LPRINT CHR$(30)
40 RETURN
41 DATA 136,136,136,136,136,136,136,136,136,136
42 DATA 166,239,201,201,201,201,201,201,251,178
43 DATA 128,255,255,137,137,137,153,185,239,198
44 DATA 129,129,129,129,255,255,129,129,129,129
45 DATA 255,255,255,255,255,255,255,255,255,255
46 DATA 190,255,225,241,217,205,199,195,255,190
47 DATA 182,255,201,201,201,201,201,201,255,182
48 DATA -1
```

Listing 12-6. Pattern Generation Subroutine

printing more than one graphics line, which would be necessary if the matrix was over seven dots high. For this reason we adopted the scheme used on early Radio Shack printers of not having descenders on lowercase letters. (The descender is the portion of the character that goes below the base line of the character.) We have a character set of 96 characters here with a provision for up to 256 characters. You can easily add your own, or modify the existing characters. We'll show you how. By the way, there's no reason that this character set can't be used for any symbol that can be defined in a six-by-seven matrix. Several characters can be combined to produce much larger designs and patterns, also.

You don't have to worry about how the subroutine works, but there are a few cautions we must make regarding its use. First of all, you can't use two-letter variable names starting with Z, such as ZZ, ZA, or ZA$, in your own program. Secondly, if you have DATA statements in your own program, you cannot use the value 999999 in any of the DATA values. Also, make certain your DATA statements come first, before those in the 10000 area. (The pro-

gram assumes you might have DATA statements and searches for its data.)

To show you how the program works to generate Figure 12-25, here's the earlier code, modified for the subroutine call.

```
100 LPRINT CHR$(18);CHR$(28);CHR$(XXX);CHR$(XXX);
110 LPRINT CHR$(255): ZZ$ = "TEXT": GOSUB 10000
120 LPRINT CHR$(255);CHR$(28);CHR$(XXX);CHR$(XXX)
```

REDEFINING THE CHARACTER SET OR DESIGNING YOUR OWN CHARACTERS

You can substitute any characters in place of the ones given above, or add to the character set in the codes 128 through 255. To do this, determine what the code for the character will be. Suppose you wanted to add the shape shown in Figure 12-27 as code 128. (This is the secret symbol for the Benevolent and Protective Order of Ancient Programmers (BPOAP), by the way.)

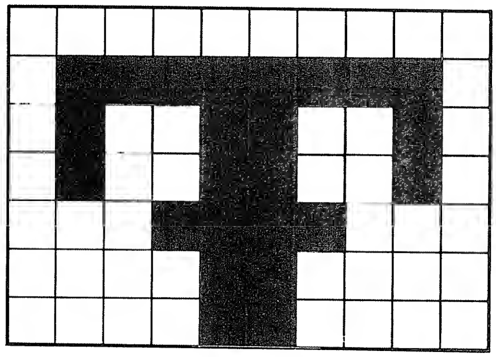


Figure 12-27. Custom Character

First of all, draw the figure out on a seven by ten grid, as shown in the figure. Now encode each of the six columns of the figure the same way you'd specify a graphics data column. The topmost bit is 1, the next 2, the next 4, the next 8, the next 16, the next 32, and the last is 64. Add 128 to the final result. You now have ten values, in this case 0, 14, 2, 18, 254, 254, 18, 2, 14, and 0. Find the 128th set of values in the data statements and replace the zeroes you find there with the ten values. Now anytime you call the subroutine with a CHR$(128), you'll get the character you defined.

```
100 ZZ$ = "This is the Symbol " + CHR$(128) + " and it looks
nice"
110 GOSUB 10000
```

DEFINING LARGER BLOCKS

You can use several of the characters to make a larger symbol. Suppose you want to make the four-element symbol shown in Figure 12-28. Define four sets of ten values, as shown in the figure and then replace codes 129 through 132 as the four parts of the figure. To print out the figure, use two graphics lines:

```
100 ZZ$ = CHR$(129) + CHR$(130): GOSUB 10000
```

```
110 LPRINT
120 ZZ$ = CHR$(131) + CHR$(132): GOSUB 10000
```

Note that the subroutine never prints the line (never sends a carriage return code). It's always up to your own code to do that.

There are 128 undefined blocks of code which you can define in any way you wish. This technique can be used to draw diagrams which utilize the same types of symbols over and over, such as electronic schematic or logic diagrams, plumbing or electrical wiring, or others, as shown in Figure 12-29.

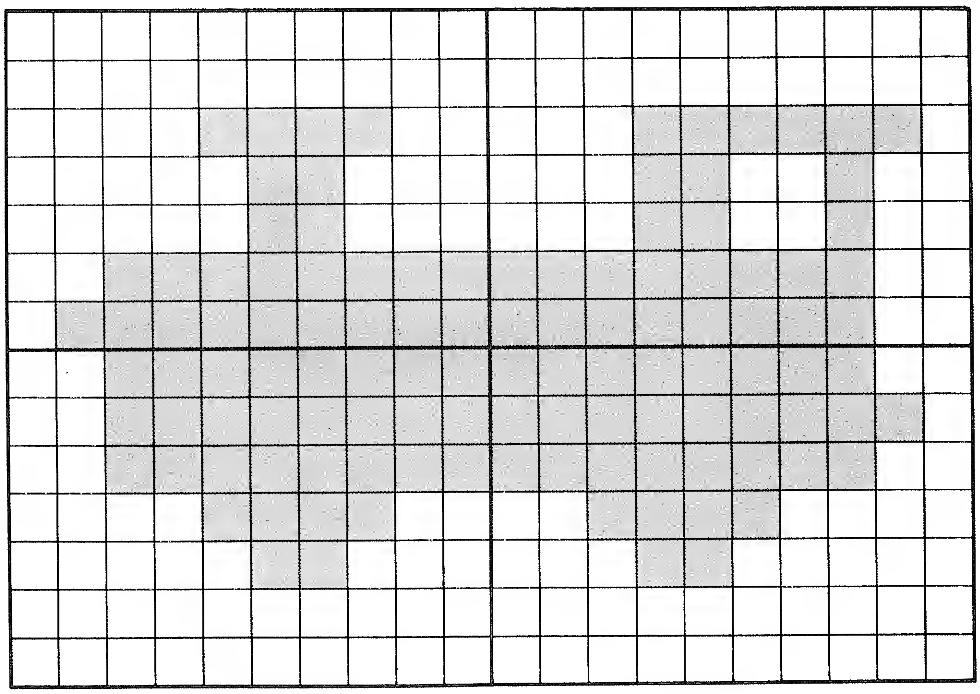


Figure 12-28. Four-Element Character

DEFINING FINER CHARACTERS

No, the title of this section doesn't mean finer characters as in "Mighty Fine," but finer in terms of *resolution*. Five-by-seven characters are fine for utilitarian printing in graphics mode, but perhaps you'd like to design your own font or typeface. We'll use a 24-by-16 matrix to do this, primarily because that's the vertical resolution of the DMP-2100, the most dense graphics printer in the Radio Shack line. The technique, however, will work with any Radio Shack printer. Printers other than the DMP-2100 will have to print two graphics lines for each character, making the minimum character size about 0.2 inch high and the number of characters per inch about five.

A grid for drawing such characters is shown in Figure 12-30, complete with stylized "T" character. There are two parts to the grid. Printers other than the DMP-2100 can use the grid with 14 rows and 16 columns. The DMP-2100 can use the grid with 16 rows and 16 columns.

To define a character, draw it on the grid and then fill in the dots that most accurately represent the character. If you are defining a character for the

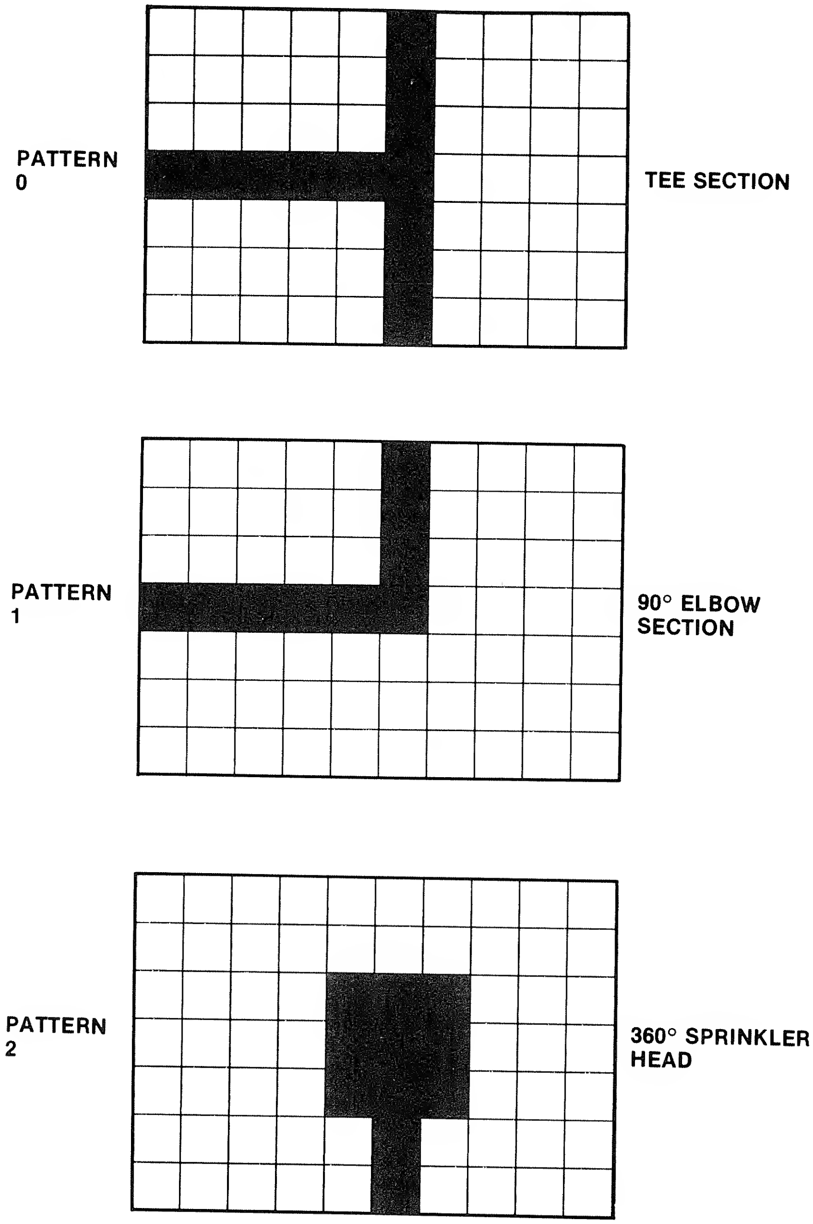


Figure 12-29. Custom Character Set

DMP-2100, read "DMP-2100 High-Resolution Graphics," and remember that you should not use adjacent pins in the same column.

The next step is to convert the character columns to CHR$() values. If you are not using the DMP-2100, the first seven dots define the first CHR$() value, and the dots in the second seven dots define the second CHR$() value for the column. If you are using the DMP-2100, the first eight points define the first CHR$() value and the next eight points define the second CHR$() value. The third CHR$() is always CHR$(0) to provide spacing for alternate rows of characters.

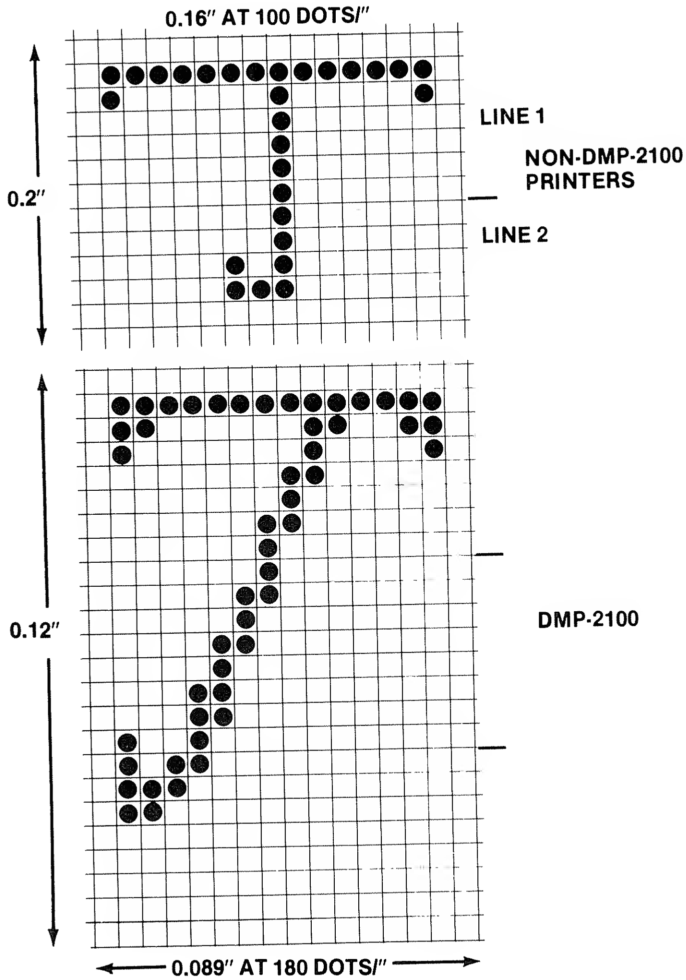


Figure 12-30. Fine Characters

To convert for non DMP-2100 printers, the dots are weighted from top to bottom in values of 1, 2, 4, 8, 16, 32, and 64. If you are using the DMP-2100, the dots are weighted from top to bottom in values of 128, 64, 32, 16, 8, 4, 2, and 1. Add up the dots for each group, and add 128 for the non DMP-2100 case. The converted values for both the DMP-2100 and other printers are shown in Figure 12-31.

At this point, if you're using a DMP-2100, follow the printing escape sequence described in "DMP-2100 High-Resolution Graphics." For other printers you must use the printer escape sequence described in "Basics of Graphics Mode," on two separate graphics lines. The top half of the character is printed first by sending 16 columns of data. The bottom half of the character is then printed in the next graphic line, at the same horizontal position as the top half. This means that all upper halves of text must be printed first, followed by all lower halves. It will take some BASIC overhead to make certain that the characters are positioned correctly. Listing 12-7 shows two short programs to print out the string of the characters we've defined above, and Figure 12-32 shows the result.

Printer Hint

"SHOOTING DOWN"

One favorite trick used by commercial illustrators and graphics arts people for making graphics and printing look good is make it big and shoot it down. Make your print large and then have an offset-printing plate made in a smaller size. It's commonplace for a print shop to reduce your camera-ready copy down to any size you desire. You might consider designing some of your own type fonts four or five times oversize with some of the techniques in this section, then reducing the copy for letterheads, logos, or other copy.

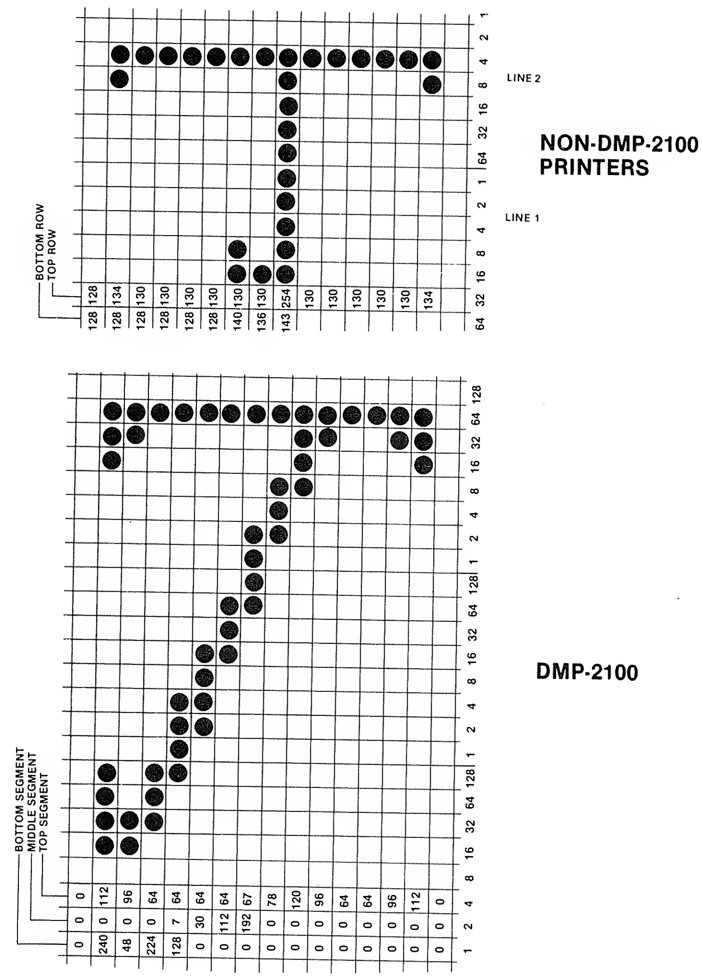


Figure 12-31. Converting Values for Fine Characters

COLOR PRINTING ON THE CGP-220

The CGP-220 is a dot-matrix ink-jet printer capable of printing dots in black, red, green, or blue, or different combinations of red, green, and blue. If you're not printing in color, the CGP-220 behaves pretty much like other dot-matrix printers—it prints text and standard seven-column graphics in 640 dot columns across the paper. When you're using the CGP-220 in the Color Scan mode, however, the printer requires a set of commands different from anything we've described so far.

Figure 12-32. Printing a Fine Character

PRINTING TEXT IN COLORS

The basic command for changing colors in the printer uses the escape sequence CHR$(27);CHR$(84);CHR$(C), where C is a color code of 48 through 55:

48 = Black
49 = Red
50 = Green
51 = Yellow

```
100 'NON-2100 PRINT GRAPHICS PROGRAM
110 LPRINT CHR$(27);CHR$(20)
120 LPRINT CHR$(18)
130 READ A
140 IF A=-1 THEN 170
150 LPRINT CHR$(A);
160 GOTO 130
170 LPRINT CHR$(30)
180 END
190 DATA 128,134,130,130,130,130,130,130,254,130,130,130,130,130,134,13
200 DATA 128,128,128,128,128,128,140,136,143,13
210 DATA -1
```

```
100 'DMP 2100 PRINT PROGRAM
110 LPRINT CHR$(27);CHR$(73);CHR$(0);CHR$(16);
120 FOR I=1 TO 48
130 READ A
140 LPRINT CHR$(A);
150 NEXT I
160 END
170 DATA 0,0,0,112,0,240,96,0,48,64,0,224,64,7,128,64,30,0,64,112,0,67,192,0
180 DATA 78,0,0,120,0,0,96,0,0,64,0,0,64,0,0,96,0,0,112,0,0,0,0,0
```

Listing 12-7. Fine Character Programs

52 = Blue (Cyan)
53 = Magenta
54 = Violet
55 = White (no print)

Changing the color code changes the color for any text printing, so you can switch back and forth between these colors to print text in different colors for highlighting:

LPRINT "This is regular text";CHR$(27);CHR$(84);CHR$(49);
" and this is text in RED";CHR$(27);CHR$(84);CHR$(51);
" and YELLOW"

THE COLOR SCAN MODE

The color mode in the CGP-220 is called the *color-scan mode*. This mode uses a bit-encoding scheme reminiscent of standard graphics, but in spite of what the operating manual tells you, is really quite different!

In this mode, the print line is divided into 80 *dot-rows*, as shown in Figure 12-33. Each dot row controls eight dots across the row. There's a total of 640 dots across any print line, so it takes 80 dot rows to control each print line.

The dot-row is the same as one of the rows used to make up a row of text characters or a row of graphics characters, as you might suspect. The spacing between color-scan rows is the same as a one-to-one ratio in the other modes—about 0.08 inch for seven vertical dots, or about 80 dots per vertical inch, as shown in the figure.

The CGP-220 normally prints from 1 to 80 dot-rows, although more than 80 dot-rows can be printed. Usually you'd print all 80 dot-rows, or 640 dots across the color line. Because the printer uses three color sources, red, green, and blue, and because you can mix the colors in any dot position, you must specify a red-green-blue color for each dot!

The escape sequence for doing this is CHR$(27);CHR$(67);CHR$(LL); CHR$(. . .) . . . CHR$(. . .). The first two values mark this as the sequence

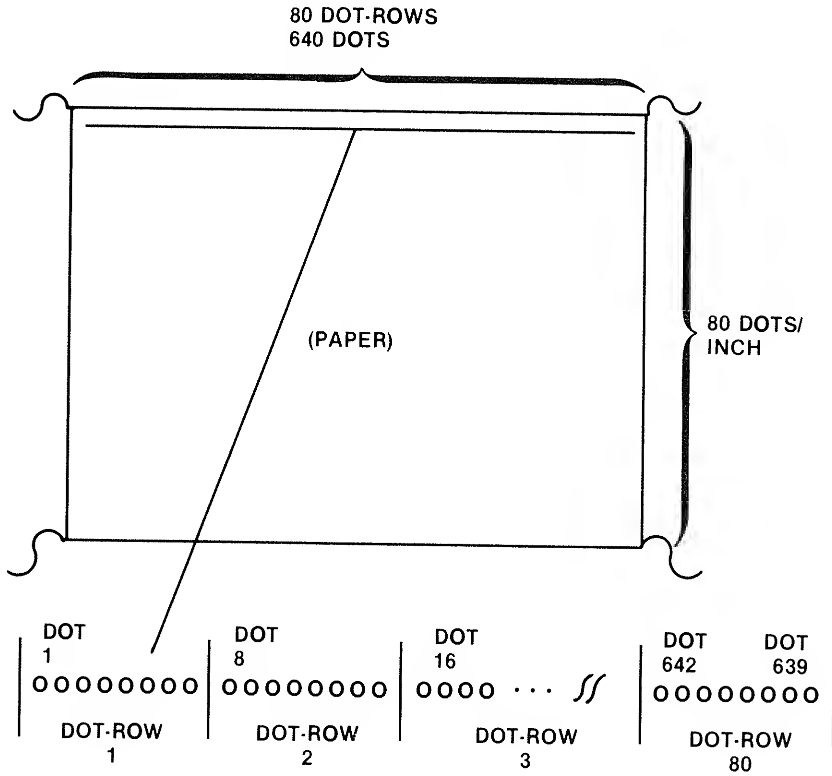


Figure 12-33. CGP-220 Dot Rows

to set color-scan mode. The next value, CHR$(LL), tells the printer how many dot-rows you want to print. If you want to print 100 dots along the line, for example, the LL value would be 13—100/8 is 12.5, but you can't print a partial dot-row. If you want to print 80 dot-rows, or all 640 dots along the line, you'd specify 80 for LL.

The number of values following the first three CHR$ values depends upon the number of dot-rows you specify. There will be three times the number of dot-rows you specify in the CHR$(LL) value—*three values for each dot-row* to specify red, green, and blue for each dot. If you print 80 dot-rows (640 dots), you'd have 80 times 3 or 240 values following.

The values following are in a definite order: First come all of the values for red, next the green values, and finally the blue values.

Suppose you wanted to print ten dot-rows, or 80 dots, as shown in Figure 12-34. You'd have a total of ten red values, ten green values, and ten blue values. It helps if you think of the values arranged to correspond to the dot rows, as shown in the figure.

What's in the value? Each value contains a code value for the eight dots per dot-row that it represents. The first red value contains on-off codes for the first eight dots. The next red value contains on-off codes for the next eight dots, and so forth. The first blue value contains on-off codes for the first eight dots. The next blue value contains on-off codes for the second eight dots, and so forth. The green values are similar.

For each dot-row, the dots have weighted codes similar to those in the standard graphics, as shown in Figure 12-35. The left dot has a weight of 128;

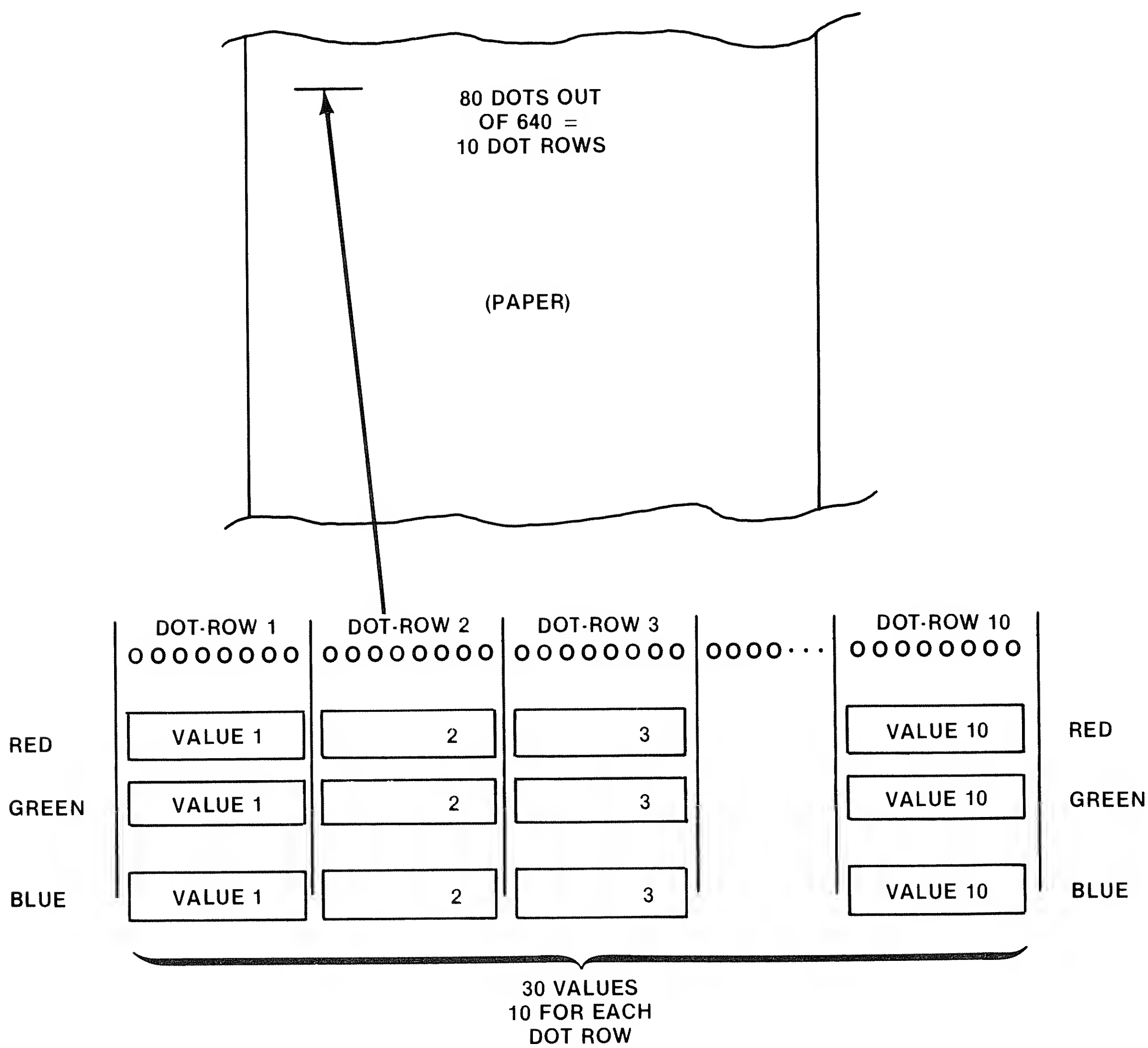


Figure 12-34. Color Values in Dot Rows

the next dot, a weight of 64, the next 32, the next 16, the next 8, the next 4, the next 2, and the last 1. To find the code for the dot-row, add up all of the codes for those dots which should be on. The result is the CHR$ value.

Suppose you wanted to print the first, second-to-last, and last dot of a dot-row. You'd have 128 + 2 + 1 or 131 for the CHR$ value. The value for each dot-row is computed separately from any other. In our example above, there are ten values for each color, representing 80 dot columns.

This encoding process is repeated for each of the three colors. In the example above we'd have 30 values, the first ten for the red colors in the dot-rows, the next ten for the green colors in the same dot-rows, and the next ten for the blue colors in the same dot rows. The most important thing to remember is that there are three ink jets, and that red, green, and blue ink, or any combination, can be squirted onto a dot position at the same time.

What happens when you mix colors? As in the red/green/blue mixing of a color

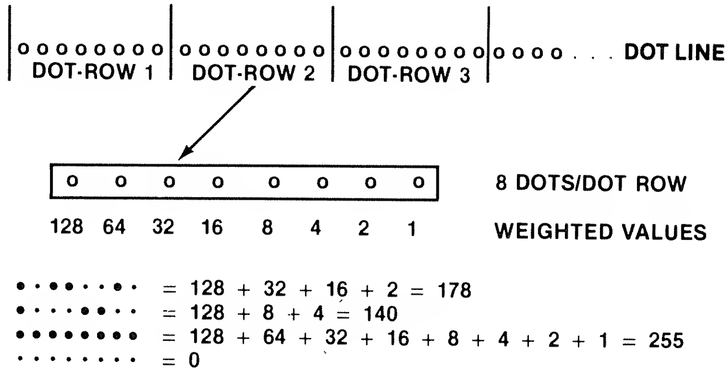


Figure 12-35. Dot Weights in Dot Rows

television tube, the three basic colors can be mixed to create other colors.

To see how this process works, look at the following program. It prints 100 dot lines across the paper, or about 100/80, 1.25 inches. The outer loop is controlled by variable K, which prints 100 complete rows. For each row, the CHR$(27);CHR(67);CHR$(80); print 80 dot-columns, or 640 dots across the row. With 80 dot-rows there must be 240 corresponding values, 80 for each of the three colors.

The 80 values are produced in three segments, each controlled by a FOR I = 1 TO 10 loop that LPRINTs eight values. What colors can we expect? This program produces all possible combinations of colors in vertical bands. The sequence goes like this: black (0,0,0), blue (0,0,255), green (0,255,0), violet (0,255,255), red (255,0,0), magenta (255,0,255), yellow (255,255,0), and white (255,255,255). The 255 value turns on (prints) all dots in a dot-row for the color, and the 0 value turns all dots off (doesn't print) the color. However, the (255, 255, 255) notation to turn all dots *on*, will result in a "no print."

```
100 FOR K=1 TO 100
110 LPRINT CHR$(27);CHR$(67);CHR$(80);
120 FOR I=1 TO 10
130 LPRINT CHR$(0);CHR$(0);CHR$(0);CHR$(0);CHR$(255);
CHR$(255);CHR$(255);CHR$(255);
140 NEXT I
150 FOR I=1 TO 10
160 LPRINT CHR$(0);CHR$(0);CHR$(255);CHR$(255);CHR$(0);
CHR$(0);CHR$(255);CHR$(255);
170 NEXT I
180 FOR I=1 TO 10
190 LPRINT CHR$(0);CHR$(255);CHR$(0);CHR$(255);CHR$(0);
CHR$(255);CHR$(0);CHR$(255);
200 NEXT I
210 NEXT K
```

Printer Hint

MORE ABOUT THE CGP-220

The color-scan mode of the CGP-220 uses a one-to-one dot pitch. This adjusts the height-to-width-ratio dot positioning so that the dot density is the same horizontally and vertically. This is compatible with the Color Computer screen, where the dot density vertically and horizontally is one-to-one in a properly adjusted television. (There are 256 dots horizontally to 192 dots vertically in high-resolution mode in the Color Computer, but the screen proportion is four units wide by three units high.)

The one-to-one ratio can also be used in text mode by sending the escape sequence CHR$(27);CHR$(78). In this mode, the characters will be squashed down on the lines, but the line spacing will remain the same—six lines per inch. The normal three-to-four proportion can be reset by the escape sequence CHR$(27);CHR$(80).

APPENDIX

Decimal	Binary	Hexa-decimal
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F
16	00010000	10
17	00010001	11
18	00010010	12
19	00010011	13
20	00010100	14
21	00010101	15
22	00010110	16
23	00010111	17
24	00011000	18
25	00011001	19
26	00011010	1A
27	00011011	1B
28	00011100	1C
29	00011101	1D
30	00011110	1E
31	00011111	1F
32	00100000	20
33	00100001	21
34	00100010	22
35	00100011	23
36	00100100	24
37	00100101	25
38	00100110	26
39	00100111	27
40	00101000	28
41	00101001	29
42	00101010	2A
43	00101011	2B
44	00101100	2C
45	00101101	2D
46	00101110	2E
47	00101111	2F
48	00110000	30
49	00110001	31
50	00110010	32
51	00110011	33
52	00110100	34
53	00110101	35
54	00110110	36
55	00110111	37
56	00111000	38
57	00111001	39
58	00111010	3A
59	00111011	3B
60	00111100	3C
61	00111101	3D
62	00111110	3E
63	00111111	3F
64	01000000	40
65	01000001	41
66	01000010	42
67	01000011	43
68	01000100	44
69	01000101	45
70	01000110	46
71	01000111	47
72	01001000	48
73	01001001	49
74	01001010	4A
75	01001011	4B
76	01001100	4C
77	01001101	4D
78	01001110	4E
79	01001111	4F
80	01010000	50
81	01010001	51
82	01010010	52
83	01010011	53
84	01010100	54
85	01010101	55
86	01010110	56
87	01010111	57
88	01011000	58
89	01011001	59
90	01011010	5A
91	01011011	5B
92	01011100	5C
93	01011101	5D
94	01011110	5E
95	01011111	5F
96	01100000	60
97	01100001	61
98	01100010	62
99	01100011	63
100	01100100	64
101	01100101	65
102	01100110	66
103	01100111	67
104	01101000	68
105	01101001	69
106	01101010	6A
107	01101011	6B
108	01101100	6C
109	01101101	6D
110	01101110	6E
111	01101111	6F
112	01110000	70
113	01110001	71
114	01110010	72
115	01110011	73
116	01110100	74
117	01110101	75
118	01110110	76
119	01110111	77
120	01111000	78
121	01111001	79
122	01111010	7A
123	01111011	7B
124	01111100	7C
125	01111101	7D
126	01111110	7E
127	01111111	7F
128	10000000	80
129	10000001	81
130	10000010	82
131	10000011	83
132	10000100	84
133	10000101	85
134	10000110	86

Decimal	Binary	Hexa-decimal
135	10000111	87
136	10001000	88
137	10001001	89
138	10001010	8A
139	10001011	8B
140	10001100	8C
141	10001101	8D
142	10001110	8E
143	10001111	8F
144	10010000	90
145	10010001	91
146	10010010	92
147	10010011	93
148	10010100	94
149	10010101	95
150	10010110	96
151	10010111	97
152	10011000	98
153	10011001	99
154	10011010	9A
155	10011011	9B
156	10011100	9C
157	10011101	9D
158	10011110	9E
159	10011111	9F
160	10100000	A0
161	10100001	A1
162	10100010	A2
163	10100011	A3
164	10100100	A4
165	10100101	A5
166	10100110	A6
167	10100111	A7
168	10101000	A8
169	10101001	A9
170	10101010	AA
171	10101011	AB
172	10101100	AC
173	10101101	AD
174	10101110	AE
175	10101111	AF
176	10110000	B0
177	10110001	B1
178	10110010	B2
179	10110011	B3
180	10110100	B4
181	10110101	B5
182	10110110	B6
183	10110111	B7
184	10111000	B8
185	10111001	B9
186	10111010	BA
187	10111011	BB
188	10111100	BC
189	10111101	BD
190	10111110	BE
191	10111111	BF
192	11000000	C0
193	11000001	C1
194	11000010	C2
195	11000011	C3
196	11000100	C4
197	11000101	C5
198	11000110	C6
199	11000111	C7
200	11001000	C8
201	11001001	C9
202	11001010	CA
203	11001011	CB
204	11001100	CC
205	11001101	CD
206	11001110	CE
207	11001111	CF
208	11010000	D0
209	11010001	D1
210	11010010	D2
211	11010011	D3
212	11010100	D4
213	11010101	D5
214	11010110	D6
215	11010111	D7
216	11011000	D8
217	11011001	D9
218	11011010	DA
219	11011011	DB
220	11011100	DC
221	11011101	DD
222	11011110	DE
223	11011111	DF
224	11100000	E0
225	11100001	E1
226	11100010	E2
227	11100011	E3
228	11100100	E4
229	11100101	E5
230	11100110	E6
231	11100111	E7
232	11101000	E8
233	11101001	E9
234	11101010	EA
235	11101011	EB
236	11101100	EC
237	11101101	ED
238	11101110	EE
239	11101111	EF
240	11110000	F0
241	11110001	F1
242	11110010	F2
243	11110011	F3
244	11110100	F4
245	11110101	F5
246	11110110	F6
247	11110111	F7
248	11111000	F8
249	11111001	F9
250	11111010	FA
251	11111011	FB
252	11111100	FC
253	11111101	FD
254	11111110	FE
255	11111111	FF

INDEX

HOW TO USE YOUR RADIO SHACK PRINTER

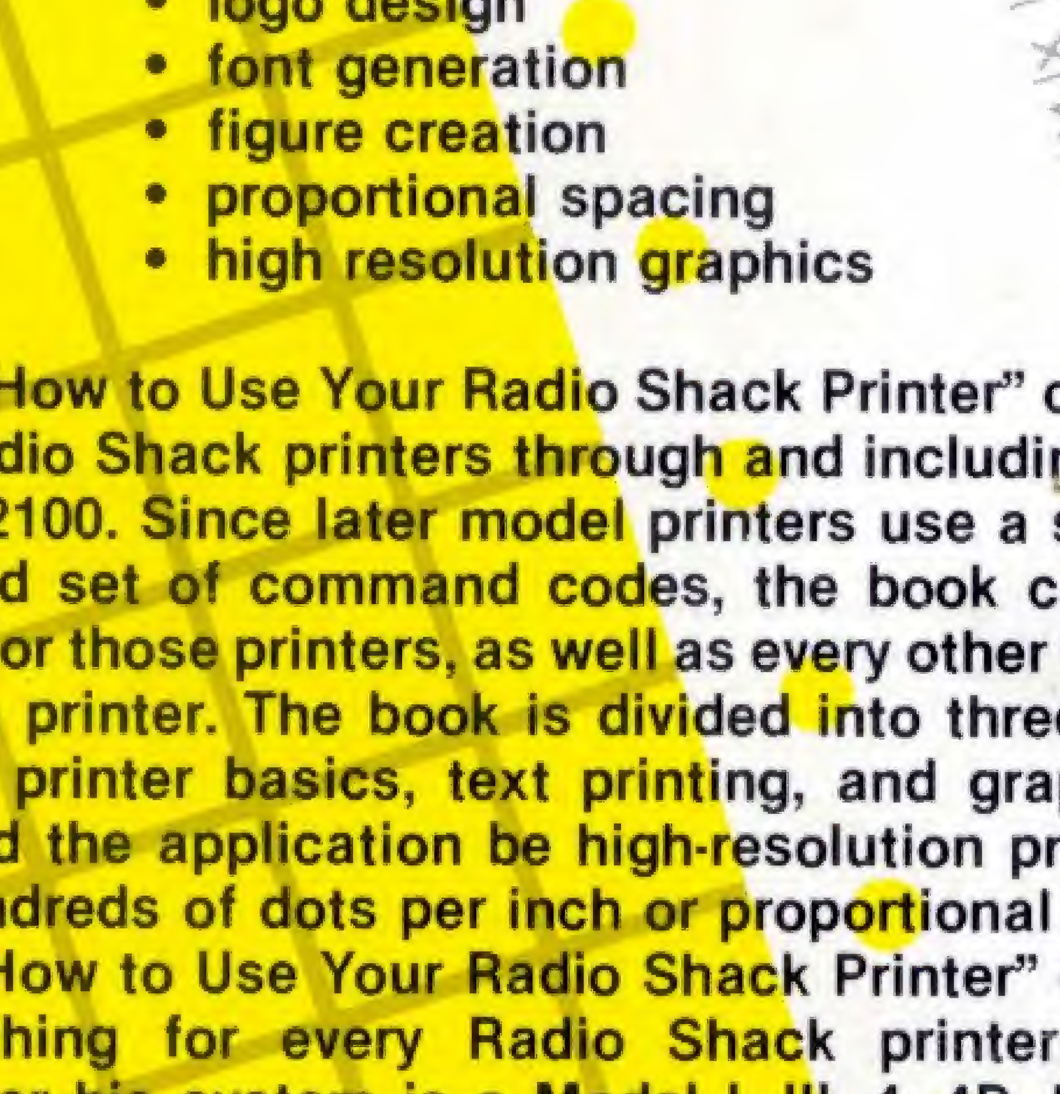
Cat. No.
26-1242

"...getting more out of your Radio Shack Printer"

This book will allow any Radio Shack computer user to tap the hidden resources of his Radio Shack printer, and lead him step by step through a variety of functions such as:

- forms printing
- logo design
- font generation
- figure creation
- proportional spacing
- high resolution graphics

"How to Use Your Radio Shack Printer" covers all Radio Shack printers through and including the DMP-2100. Since later model printers use a standardized set of command codes, the book can be used for those printers, as well as every other Radio Shack printer. The book is divided into three sections: printer basics, text printing, and graphics. Should the application be high-resolution printing at hundreds of dots per inch or proportional spacing, "How to Use Your Radio Shack Printer" offers something for every Radio Shack printer user whether his system is a Model I, III, 4, 4P, Model 100, Color Computer, MC-10, Tandy 2000, Tandy 1000 or Tandy 1200.



ABOUT THE AUTHOR: **William Barden, Jr.**, is one of the country's most widely read computer book authors. His Radio Shack books include "TRS-80 Assembly Language Programming," "Programming Techniques for Level II BASIC," "Business Applications Programming Guide," "Color Computer Graphics," "Color Computer Assembly Language Programming," and "TRS-80 Pocket BASIC Handbook." In addition to the Radio Shack books, Barden has developed several software projects for Radio Shack, including "Assembly Language Tutor for the Model I/III" and "Assembly Language Tutor for the Color Computer." He has authored over 27 books in the computer area, including "Z-80 Microcomputer Handbook," "How to Program Microcomputers," "Z-80 Microcomputer Design Projects," "IBM PC Programs," and "IBM PC and PCjr Encyclopedia."

0-915391-09-0